POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INTERNETOWYCH (IST)

PRACA DYPLOMOWA

MAGISTERSKA

Projekt i implementacja wielowarstwowej rozproszonej aplikacji internetowej działającej na platformie Java EE, wspierającej

działalność przedsiębiorstwa handlowego

Design and implementation of a multi-layered distributed web application operating on the Java EE platform, supporting the operation of a commercial enterprise.

AUTOR:

Michał Hawełka

PROWADZĄCY PRACĘ:

dr inż. Zofia Kruczkiewicz

OCENA PRACY:

WROCŁAW 2018

**Spis treści**

[Spis tabel 7](#_Toc519797505)

[Spis listingów 8](#_Toc519797506)

[Skróty 9](#_Toc519797507)

[1. Wstęp 10](#_Toc519797508)

[1.1. Cel pracy 10](#_Toc519797509)

[1.2. Zakres pracy 11](#_Toc519797510)

[2. Technologie używane do budowy aplikacji internetowych 12](#_Toc519797511)

[2.1. Wielowarstwowa rozproszona aplikacja internetowa 12](#_Toc519797512)

[2.2. Warstwa klienta 13](#_Toc519797513)

[2.3. Warstwa sieciowa / warstwa prezentacji 13](#_Toc519797514)

[2.3.1. Java Server Faces 14](#_Toc519797515)

[2.3.2. Model View Controller 14](#_Toc519797516)

[2.4. Warstwa biznesowa 15](#_Toc519797517)

[2.4.1. Funkcjonalności Java EE 16](#_Toc519797518)

[2.5. Warstwa integracji 19](#_Toc519797519)

[2.5.1. Java Messaging Service 19](#_Toc519797520)

[2.5.2. Java Persistence API 21](#_Toc519797521)

[2.5.3. Hibernate 21](#_Toc519797522)

[2.6. Warstwa EIS 22](#_Toc519797523)

[2.6.1. PostgreSQL 22](#_Toc519797524)

[3. Opracowanie projektu wielowarstwowej rozproszonej aplikacji internetowej 24](#_Toc519797525)

[3.1. Opis świata rzeczywistego 24](#_Toc519797526)

[3.1.1. Opis zasobów ludzkich 24](#_Toc519797527)

[3.1.2. Dane techniczne 24](#_Toc519797528)

[3.2. Wymagania funkcjonalne 25](#_Toc519797529)

[3.3. Wymagania niefunkcjonalne 25](#_Toc519797530)

[3.4. Przypadki użycia 25](#_Toc519797531)

[3.4.1. Wyszukaj zamówienia użytkownika 27](#_Toc519797532)

[3.4.2. Dodaj przedmiot 27](#_Toc519797533)

[3.4.3. Edytuj/Usuń egzemplarz 28](#_Toc519797534)

[3.4.3. Anuluj zamówienie 30](#_Toc519797535)

[3.5. Model danych 31](#_Toc519797536)

[3.5.1. ItemName (Nazwa Przedmiotu) 31](#_Toc519797537)

[3.5.2. Item (Egzemplarz) 31](#_Toc519797538)

[3.5.3. Basket (Koszyk) 32](#_Toc519797539)

[3.5.4. Order (Zamówienie) 32](#_Toc519797540)

[3.5.5. User (Użytkownik) i encje powiązane 32](#_Toc519797541)

[4. Szczegóły projektowe 33](#_Toc519797542)

[4.1. Diagramy sekwencji 33](#_Toc519797543)

[4.1.1. Wyszukaj zamówienia użytkownika 33](#_Toc519797544)

[4.1.2. Dodaj przedmiot 33](#_Toc519797545)

[4.1.3. Edytuj egzemplarz 34](#_Toc519797546)

[4.1.4. Usuń egzemplarz 35](#_Toc519797547)

[4.1.5. Anuluj zamówienie 36](#_Toc519797548)

[4.2. Interfejs użytkownika 37](#_Toc519797549)

[5. Szczegóły implementacyjne 40](#_Toc519797550)

[5.1. Architektura systemu 40](#_Toc519797551)

[5.2. Rozproszoność systemu 40](#_Toc519797552)

[5.3. Bezpieczeństwo 44](#_Toc519797553)

[5.3.1. Trwała subskrypcja (durable subsription) 44](#_Toc519797554)

[5.4. Interfejs użytkownika 45](#_Toc519797555)

[6. Pomiary wydajności i analiza wyników 47](#_Toc519797556)

[6.1. JMeter 47](#_Toc519797557)

[6.2. Środowiska testowe 48](#_Toc519797558)

[6.2.1. Dodanie przedmiotu 49](#_Toc519797559)

[6.2.2. Edycja egzemplarza 51](#_Toc519797560)

[6.2.3. Wyszukaj zamówienia użytkownika 52](#_Toc519797561)

[6.2.4. Usuń przedmiot 54](#_Toc519797562)

[6.3. Otrzymane wyniki 55](#_Toc519797564)

[6.4. Analiza wyników 60](#_Toc519797565)

[7. Podsumowanie i wnioski 62](#_Toc519797566)

[Literatura 63](#_Toc519797567)

**Spis rysunków**

[*Rys. 1. Schemat warstw wielowarstwowej aplikacji internetowej [2].* 13](#_Toc519797568)

[*Rys. 2. Logo Java Server Faces [3].* 14](#_Toc519797569)

[*Rys. 3. Uproszczony schemat MVC [4].* 15](#_Toc519797570)

[*Rys. 4. Przykładowa architektura systemu na platformie Java EE [5]* 16](#_Toc519797571)

[*Rys. 5. Logo Java EE [5]* 17](#_Toc519797572)

[*Rys. 6. Cykl życia bezstanowego sesyjnego EJB* 18](#_Toc519797573)

[*Rys. 7. Cykl życia stanowego sesyjnego EJB [6]* 18](#_Toc519797574)

[*Rys. 8. Architektura JMS API [5]* 20](#_Toc519797575)

[*Rys. 9. Architektura Point-to-Point [5]* 20](#_Toc519797576)

[*Rys. 10. Architektura Publish/Subscribe [5]* 21](#_Toc519797577)

[*Rys. 11. Logo Hibernate [8]* 22](#_Toc519797578)

[*Rys. 12. Logo PostgreSQL* 23](#_Toc519797579)

[*Rys. 13. Diagram przypadków użycia.* 26](#_Toc519797580)

[*Rys. 14. Model danych sklepu z multimediami* 31](#_Toc519797581)

[*Rys. 15. Diagram sekwencji "Wyszukaj zamówienia użytkownika"* 33](#_Toc519797582)

[*Rys. 16. Diagram sekwencji PU "Dodaj przedmiot" - część przed wysłaniem wiadomości JMS* 34](#_Toc519797583)

[*Rys. 17. Diagram sekwencji PU "Dodaj przedmiot" - część po odebraniu wiadomości JMS* 34](#_Toc519797584)

[*Rys. 18. Diagram sekwencji PU "Edytuj egzemplarz" - część przed wysłaniem wiadomości JMS* 35](#_Toc519797585)

[*Rys. 19. Diagram sekwencji PU "Edytuj egzemplarz" - część po wysłaniu wiadomości JMS* 35](#_Toc519797586)

[*Rys. 20. Diagram sekwencji PU "Usuń egzemplarz" - część przed wysłaniem wiadomości JMS* 36](#_Toc519797587)

[*Rys. 21. Diagram sekwencji PU "Usuń egzemplarz" - część po odebraniu wiadomości JMS* 36](#_Toc519797588)

[*Rys. 22. Diagram sekwencji PU "Anuluj zamówienie" - część przed wysłaniem wiadomości JMS* 37](#_Toc519797589)

[*Rys. 23. Diagram sekwencji PU "Anuluj zamówienie" - część po odebraniu wiadomości JMS* 37](#_Toc519797590)

[*Rys. 24. Ekran dodawania przedmiotu przez pracownika* 38](#_Toc519797591)

[*Rys. 25. Ekran wyszukiwania egzemplarzy przez pracownika.* 39](#_Toc519797592)

[*Rys. 26. Ekran koszyka widziany z poziomu klienta.* 39](#_Toc519797593)

[*Rys. 27. Topologia systemu.* 40](#_Toc519797594)

[*Rys. 28. Konfiguracja fabryk połączeń JMS* 42](#_Toc519797595)

[*Rys. 29. Przykładowa konfiguracja fabryki połączeń.* 43](#_Toc519797596)

[Rys. 30. Konfiguracja wątków JMS 43](#_Toc519797597)

[*Rys. 31. Konfiguracja przykładowego wątku JMS.* 43](#_Toc519797598)

[*Rys. 32. Konfiguracja testowa 1.* 48](#_Toc519797599)

[*Rys. 33. Konfiguracja testowa 2.* 49](#_Toc519797600)

[*Rys. 34. Konfiguracja JMeter dla dodawania przedmiotów* 50](#_Toc519797601)

[*Rys. 35. Dodaj przedmiot - Żądanie HTTP* 50](#_Toc519797602)

[*Rys. 36. Konfiguracja JMeter dla edycji egzemplarzy* 51](#_Toc519797603)

[*Rys. 37. Edytuj egzemplarz - Żądanie HTTP* 52](#_Toc519797604)

[*Rys. 38. Konfiguracja JMeter dla wyszukiwania zamówień* 52](#_Toc519797605)

[*Rys. 39. Wyszukaj zamówienia - Żądanie http* 53](#_Toc519797606)

[*Rys. 40. Konfiguracja JMeter dla usuwania przedmiotów* 54](#_Toc519797607)

[*Rys. 41. Usuń przedmiot - Żądanie HTTP* 54](#_Toc519797608)

[*Rys. 42. Wyniki testu obciążeniowego dla przypadku Dodaj Przedmiot na konfiguracji z centralną użyciem JMS* 58](#_Toc519797609)

[*Rys. 43. Wyniki testu obciążeniowego dla przypadku Dodaj Przedmiot na konfiguracji z centralną bazą danych* 58](#_Toc519797610)

[*Rys. 44. Wyniki testu obciążeniowego dla przypadku Wyszukaj Zamówienia na konfiguracji z użyciem JMS* 59](#_Toc519797611)

[*Rys. 45. Wyniki testu obciążeniowego dla przypadku Wyszukaj Zamówienia na konfiguracji z centralną bazą danych* 59](#_Toc519797612)

# Spis tabel

**Nie można odnaleźć pozycji dla spisu ilustracji.**

# Spis listingów

**Nie można odnaleźć pozycji dla spisu ilustracji.**

# Skróty

**JEE** (ang. *Java Enterprise Edition*)

**XML** (ang. *eXtensible Markup Language*)

**SOAP** (ang. *Simple Object Access Protocol*)

**JPA** (ang. *Java Persistence API*)

**JSF** (ang. *Java Server Faces*)

**JMS** (ang. *Java Messaging System*)

**MDB** (ang. *Message-Driven Bean*)

**EJB** (ang. *Enterprise Java Bean*)

1. Wstęp

Rozproszone, wielowarstwowe aplikacje internetowe pełnią w dzisiejszych czasach olbrzymią rolę. Pozwalają one na używanie wielu różnych technologii, zapewniają skalowalność i bezpieczeństwo. Te aspekty sprawiły, że są one często wybierane przez przedsiębiorstwa handlowe, które dzięki aplikacjom wielowarstwowym są w stanie efektywnie zarządzać swoim inwentarzem, finansami czy obsługą zamówień. Każdy z wymienionych tematów jest tematem zdecydowanie rozległym i wymagającym dobrze napisanego systemu odpornego na zewnętrzne ataki, a także pozwalającego na łatwą rozbudowę.

* 1. Cel pracy

Celem tej pracy jest sprawdzenie, jakie możliwości mogą zaoferować swoim klientom przedsiębiorstwa korzystające z systemu wielowarstwowego używającego technologii Java EE. Obecnie na rynku jest dostępnych również kilka innych technologii, ale to Java EE ma za sobą już całkiem sporą historię i jest jednym z wiodących wyborów wśród systemów internetowych. Oprócz samych możliwości technologii zostały sprawdzone również kryteria wydajnościowe. Porównane zostały parametry wielowarstwowego systemu rozproszonego, a także prostej architektury ze scentralizowaną bazą danych. Zostały wykonane następujące zadania:

* Opracowanie i wykonanie projektu wielowarstwowej rozproszonej aplikacji internetowej wspierającej przedsiębiorstwo handlowe, działającej na platformie Java EE z wykorzystaniem technologii JMS
* Implementacja projektu wielowarstwowej rozproszonej aplikacji internetowej Java EE z wykorzystaniem technologii JMS
* Opracowanie metryk umożliwiających ocenę wydajności wykonanej aplikacji
* Wykonanie pomiarów wydajności wykonanej aplikacji
* Analiza wyników i ocena wydajności
  1. Zakres pracy

W rozdziale drugim krótko przedstawione zostały technologie używane do budowy aplikacji internetowych. Skupiono się głównie na platformie Java EE. W dalszej części rozdziału wyszczególniono również najczęściej stosowane technologie do budowy warstwy prezentacji, warstwy logiki biznesowej czy warstwy integracji (na przykład Java Persistence API czy Java Messaging Service).

W trzecim rozdziale opisany został proces opracowania koncepcji systemu – od modelu biznesowego, przez wymagania, jakie stawiane są przed systemami internetowymi, aż do opracowania modelu konceptualnego. Przedstawiono tam między innymi odpowiednie przypadki użycia, które powinna obsługiwać rozproszona aplikacja internetowa, która docelowo ma być wykorzystywana przez przedsiębiorstwo handlowe. Zaprezentowane zostały również przykładowe schematy bazy danych.

Kolejny rozdział to podsumowanie fazy projektowania systemu. Abstrakcja przyjęta w rozdziale poprzednim jest tutaj uściślona. Zaprezentowane zostały diagramy klas oraz dokładne diagramy sekwencji. Przedstawiono informacje związane z warstwą prezentacji. Pokazano projekty formularzy, które będą używane przez stronę klienta.

W rozdziale piątym przedstawione zostały szczegóły implementacji aplikacji. Oprócz najważniejszych fragmentów kodu źródłowego przedstawiono rozdzielenie kodu do poszczególnych warstw aplikacji. Pokazano architekturę systemu, czyli opis warstw i topologię oprogramowania. Przedstawiono sposób obsługi bezpieczeństwa w zakresie niezawodności i ochroną przed tzw. „stroną trzecią” w wykonanej aplikacji internetowej, a więc odporności na awarie czy prawa dostępu użytkowników.

Szósty rozdział zawiera opis testowania powstałego systemu - dokładny plan testów wraz z danymi wejściowymi i oczekiwanymi rezultatami. Przedstawiono zestawienia wyników testowania przy uwzględnieniu poszczególnych rodzajów systemów w zależności od użytej topologii, a także analizę wyników przeprowadzonych testów.

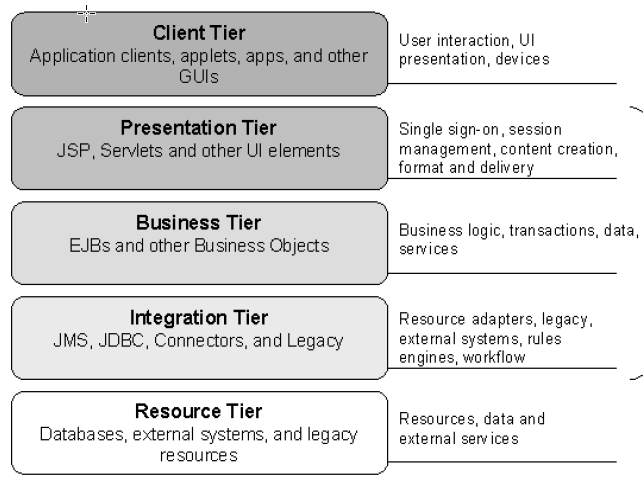
1. Technologie używane do budowy aplikacji internetowych
   1. Wielowarstwowa rozproszona aplikacja internetowa

Rozproszoną aplikacją internetową nazywany jest program, który działa na wielu komputerach w sieci. Tego typu aplikacje porozumiewają się ze sobą w celu wykonania jakiegoś zadania (na przykład utrzymywania spójności danych, wymiany informacji czy podziału odpowiedzialności)[1]. Aplikacje najczęściej są budowane na architekturze klient-serwer, gdzie n serwerów komunikuje się ze sobą. Klient odpowiada za interakcję z użytkownikiem, za wysokopoziomowy dostęp do danych, serwer z kolei odpowiada za logikę biznesową, komunikację z innymi serwerami, a także z warstwą danych (np. bazą danych)[1].

Wielowarstwowość aplikacji internetowej jest powiązana w pewien sposób z rozproszonością. Większość aplikacji rozproszonych jest aplikacjami wielowarstwowymi, ale nie wszystkie aplikacje wielowarstwowe są rozproszone. Najczęściej spotykany model aplikacji wielowarstwowej składa się z pięciu warstw:

* warstwy danych (pliki, bazy danych)
* warstwy integracji (serwisy internetowe, JMS, DAO, JPA, MDB)
* warstwy logiki biznesowej (komponenty typu Session Bean)
* warstwy sieciowej/prezentacji (serwlety, JSP, JSF)
* warstwy klienckiej (przeglądarka, aplet Java)[1].

Tego typu podział ma zapewnić aplikacji wysoką skalowalność i ułatwić jej utrzymanie w trakcie użytkowania. Poszczególne warstwy są od siebie logicznie oddzielone w celu uzyskania możliwości łatwej rozbudowy systemu czy zmniejszenia podatności na błędy[1].



*Rys. 1. Schemat warstw wielowarstwowej aplikacji internetowej [2].*

* 1. Warstwa klienta

Warstwa kliencka to punkt wejściowy do aplikacji. To tutaj użytkownik wchodzi w interakcję z systemem. Najczęściej ta warstwa aplikacji jest uruchamiana na maszynie użytkownika lub w przeglądarce internetowej. Odpowiada ona za wyświetlanie i obsługę formularzy przez użytkownika, wstępną walidację wprowadzanych danych, a także za komunikację z aplikacją umieszczoną na serwerze aplikacji[2].

* 1. Warstwa sieciowa / warstwa prezentacji

Warstwa sieciowa przechowuje wszystkie komponenty potrzebne do uruchomienia warstwy klienckiej, a także odpowiada za zarządzanie sesjami użytkowników. Popularniejszą nazwą jest warstwa prezentacji, ale w przypadku aplikacji tworzonych na platformie Java EE można mówić o warstwie sieciowej (głównie ze względu na użycie takich technologii jak serwlety Java, Java Server Pages czy Java Server Faces)[2].

* + 1. Java Server Faces

Java Server Faces (JSF) to technologia, która została opracowana w celu ułatwienia tworzenia warstwy sieciowej i warstwy prezentacji w aplikacjach sieciowych. Jej główne założenia to uniezależnienie implementacji od wyboru języka, protokołów czy urządzeń klienckich. JSF zapewnia własne urządzenie renderujące, które może być używane przez poszczególne urządzenia klienckie, dzięki czemu programista może w łatwy sposób przygotować rozwiązania dla dużej grupy odbiorców[3].



*Rys. 2. Logo Java Server Faces [3].*

Cechy Java Server Faces:

* Prostota użytkowania
* Rozdzielenie logiki i warstwy prezentacji
* Użycie wzorca MVC (Model View Controller).
  + 1. Model View Controller

Wzorzec Model-View-Controller jest już używany od dłuższego czasu, ale wraz z rozpowszechnieniem internetu zwiększyła się jego popularność, a sam wzorzec ewoluował. Założenia wzorca prezentują się następująco:

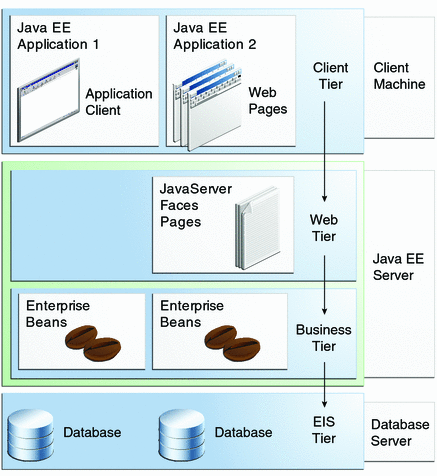
* Model – to opis lub reprezentacja danych. Nie są to właściwe dane. Model to pewnego rodzaju abstrakcja, na przykład - pozwala na używanie wielu baz danych niezależnie od ich schematu.
* Widok (View) – warstwa klienta, zawierająca formularze, które udostępnia się użytkownikowi aplikacji do wprowadzania i wyprowadzania danych użytkownika.
* Kontroler (Controller) – kontroluje przepływ informacji między modelem i widokiem. Decyduje o tym, jakie dane są wyciągane z bazy danych przez model i jakie dane są przekazywane do widoku. Implementuje też logikę biznesową. Kontroler może składać się z komponentów typu Managed Bean do tworzenia danych widoku oraz kontrolerów typu EJB do realizacji logiki biznesowej na modelu danych[4].



*Rys. 3. Uproszczony schemat MVC [4].*

* 1. Warstwa biznesowa

Warstwa biznesowa służy do przetwarzania danych biznesowych. Jest w niej zawarta cała logika biznesowa, a także takie funkcjonalności jak transakcje czy usługi internetowe. W przypadku aplikacji na platformie Java EE warstwa ta składa z komponentów typu Enterprise Java Beans (EJB)[2].



*Rys. 4. Przykładowa architektura systemu na platformie Java EE [5]*

* + 1. Funkcjonalności Java EE

Z punktu widzenia wielowarstwowej aplikacji najważniejszą funkcjonalność platformy Java EE wykonują komponenty typu Enterprise Java Beans (EJB). Są to komponenty działające po stronie serwera aplikacji, zawierające logikę biznesową. Wykorzystuje się je głównie do budowania złożonych aplikacji. Jest tak dlatego, że komponenty typu EJB pozwalają na łatwą rozbudowę programu, a także na jego rozproszenie. Same EJB są uruchamiane w tzw. kontenerach EJB, które zapewniają użytkownikowi szereg usług już na początku, bez potrzeby ich implementacji. Do tego typu usług można zaliczyć obsługę transakcji czy też bezpieczeństwa aplikacji[6].



*Rys. 5. Logo Java EE [5]*

W przypadku systemu do obsługi przedsiębiorstwa handlowego komponent EJB jest dobrym wyborem z kilku powodów:

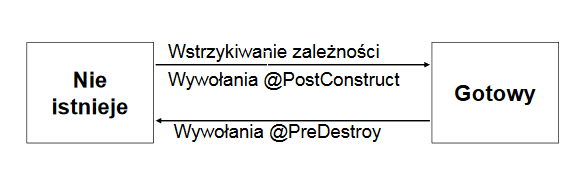
* pozwala na obsługę różnego typu klientów
* aplikacje są skalowalne
* pozwala na zaawansowane przetwarzanie transakcyjne

Komponenty typu EJB dzielą się na dwa typy komponentów – Sesyjne (Session Beans) i Komunikatowe (Message-Driven Beans). Oba typy są przydatne w projektowaniu wielowarstwowej rozproszonej aplikacji internetowej[6].

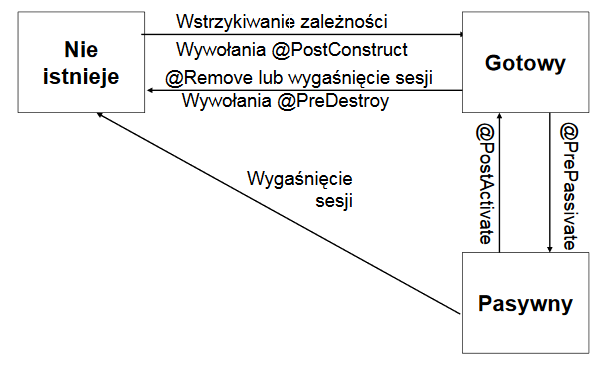
* + - 1. Session Bean

Ziarenka sesyjne mają za zadanie wykonać konkretną czynność logiki biznesowej. Nie są współdzielone, a ich stan nie wykracza poza daną sesję, a co za tym idzie, nie jest reprezentowany w warstwie danych (np. w bazie danych). Wyszczególnia się trzy podtypy ziarenek sesyjnych:

* Stanowe (Stateful) – są to komponenty, które reprezentują stan sesji z konkretnym klientem. Zapisywane są w nich informacje dotyczące działań i zapytań klienta (np. klient nie musi za każdym razem pobierać listy przedmiotów, bo ta jest zapisana w komponencie).
* Bezstanowe (Stateless) – skalowalne komponenty używane głównie do wykonywania metod logiki biznesowej (np. web services). Wyniki te są współdzielone przez wiele aplikacji klienta (internetowe i desktopowe)
* Singleton – bezstanowy komponent używany głównie do wykonywania współdzielonych metod logiki biznesowej (np. web services). Jest mniej skalowalnym komponentem w porównaniu z komponentem, typu Stateless[6]



*Rys. 6. Cykl życia bezstanowego sesyjnego EJB*



*Rys. 7. Cykl życia stanowego sesyjnego EJB [6]*

* + - 1. Message-Driven Bean

Komponenty komunikatowe pozwalają na asynchroniczną obsługę komunikatów/wiadomości. Najczęściej są to wiadomości w technologii JMS (której poświęcony jest kolejny podrozdział). Charakterystyczne dla komponentów typu Message-driven Bean (MDB) jest to, że klienci nie odwołują się do nich bezpośrednio, a właśnie za pomocą różnego rodzaju wiadomości. Główne cechy MDB to:

* Wywoływanie asynchroniczne
* Krótkotrwałe – instancje są tworzone tylko na czas obsługi żądania
* Bezstanowe
* Nie reprezentują danych w bazie danych, ale mogę ją edytować
* Mogą obsługiwać wiadomości od wielu klientów[6].
  + - 1. Wstrzykiwanie zależności (Dependency Injection)

Wstrzykiwanie zależności jest częścią funkcjonalności Java EE, zwaną w skrócie CDI (Context and Dependency Injection). Pozwala ono na efektywne i optymalne użycie zasobów przez aplikację poprzez „wstrzyknięcie” do danego komponentu referencji do zasobów, czy serwisów. Dzięki temu aplikacja nie musi ich za każdym razem tworzyć, czy wyszukiwać – są dostępne od razu[5].

* 1. Warstwa integracji

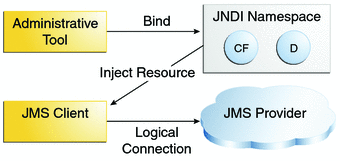
Warstwa integracji odpowiada za odpowiednie przygotowanie danych dla warstwy typu EIS (Enterprise Information System, np. baza danych, inne aplikacje – rys. 4). Jest tutaj dokonywane nie tylko mapowanie struktur danych aplikacji na bazodanowe, ale również odbywa się tutaj komunikacja z innymi serwerami czy bazami danych. Integracja może przebiegać synchronicznie lub asynchronicznie[2].

* + 1. Java Messaging Service

Java Messaging Service (JMS) to API pozwalające na tworzenie, wysyłanie, odbiór i odczyt wiadomości. Jest to tak naprawdę zestaw interfejsów ułatwiających programiście nawiązanie kontaktu z innymi implementacjami. Najważniejsze cechy JMS to:

* Luźne powiązanie (loose coupling) – API jest odporne na awarie po którejś ze stron komunikacji
* Asynchroniczność – odbiorca może otrzymywać wiadomości w dowolnym momencie, nie musi odpytywać o nie serwera
* Niezawodność, pewność – JMS zapewnia że dana wiadomość jest dostarczona do miejsca docelowego tylko raz.

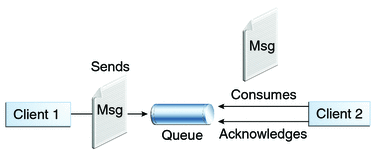
Wiadomości w JMS mogą być wysyłane do miejsc docelowych dwóch typów: kolejki (Queue) i wątku (Topic). Architektura systemu korzystającego z JMS składa się z dostawcy JMS (providera), klientów JMS, wiadomości i administrowanych obiektów[7].



*Rys. 8. Architektura JMS API [5]*

* + - 1. Kolejka (Queue)

Kolejka to przykład domeny wiadomości typu punkt-punkt (point-to-point, PTP). Charakteryzuje się ona tym, że każda wiadomość ma konkretnego adresata, a kolejka przetrzymuje wiadomości dopóki odbiorca ich nie odczyta. Odbiorca wysyła też potwierdzenie przetworzenia wiadomości[7].



*Rys. 9. Architektura Point-to-Point [5]*

* + - 1. Wątek (Topic)

Wątek reprezentuje sposób przetwarzania wiadomości zwany publikuj/subskrybuj (publish/subscribe, pub/sub). Jest to swego rodzaju tablica ogłoszeń. Wiadomości wysłane na wątek są rozsyłane do wszystkich aktualnie subskrybujących użytkowników, a więc każda wiadomość może mieć więcej niż jednego odbiorcę. Co jest ważne, nadawca i odbiorcą są połączeni zależnością czasową – jeśli konsument subskrybuje wątek po wysłaniu przez nadawcę wiadomości – wiadomości tej nie otrzyma [7].



*Rys. 10. Architektura Publish/Subscribe [5]*

* + 1. Java Persistence API

Java Persistence API (JPA) pozwala użytkownikowi na mapowanie obiektów napisanych w języku Java na encje w bazie danych. Jest to interfejs zapewniający wszystkie potrzebne operacje, aby w prosty sposób obsługiwać bazy danych bez konieczności operowania językiem SQL. Jest to interfejs technologii JPA, należy więc zdecydować się na jedną z jej implementacji. Najpopularniejszymi implementacjami są Hibernate i EclipseLink. W tej pracy skupiono się głównie na Hibernate, jako że jest to zdecydowanie implementacja o najszerszym zastosowaniu w branży [8].

* + 1. Hibernate

Hibernate jest jedną z implementacji JPA, ale nie tylko. Oprócz zapewnienia implementacji metod zawartych w Java Persistence API, Hibernate dodaje również zestaw własnych metod. Pozwala między innymi na łatwe wykorzystanie takich obiektowych zagadnień, jak dziedziczenie, polimorfizm, asocjacja czy kompozycja na poziomie bazy danych. Użytkownik nie musi się zajmować, jak te zadania rozwiązać po stronie SQL – realizują to komponenty technologii Hibernate [8]. Najważniejsze cechy Hibernate to:

* Skalowalność – struktury danych oznaczone adnotacjami Hibernate w prosty sposób można rozszerzać
* Wysoka wydajność – Hibernate zapewnia takie funkcjonalności jak lazy initialization, rozmaite strategie pobierania danych czy automatyczne wersjonowanie.
* Niezawodność – Hibernate jest już na rynku od wielu lat, co sprawiło że jest to rozwiązanie dobrze przetestowane , stabilne i wysokiej jakości[8].



*Rys. 11. Logo Hibernate [8]*

* 1. Warstwa EIS

Warstwa EIS to najczęściej baza danych. Przechowywane są tu wszystkie istotne informacje niezbędne do właściwego działania aplikacji. Bazy danych mogą być skonfigurowane na różne sposoby, w aplikacjach internetowych stosuje się na przykład:

* Centralną bazę danych – wszystkie serwery korzystają z jednej bazy danych
* Synchronizowaną bazę danych - każdy serwer ma swoją własną bazę, która jest synchronizowana przy każdej aktualizacji bazy dowolnego serwera
* Klastry danych – każdy serwer posiada swoją własną bazę, przy odczycie danych dane są zbierane z różnych serwerów [2].
  + 1. PostgreSQL

PostgreSQL jest jednym z wielu systemów zarządzania relacyjnymi bazami danych. Jest jednak jednym z trzech najpopularniejszych wolnodostępnych i darmowych rozwiązań obok MySQLa i Firebirda. To, co jednak wyróżnia PostgreSQL, to ilość funkcjonalności i rozszerzalność. Język używany w bazach tego typu jest bardzo podobny do czystego SQLa. Najważniejsze cechy systemu PostgreSQL to:

* Obsługa funkcji
* Obsługa indeksów
* Rozszerzona lista typów danych
* Możliwość definiowania obiektów przez użytkownika
* Brak maksymalnego rozmiaru bazy
* Obsługa rozszerzeń
* Łatwe zarządzanie za pomocą intuicyjnego narzędzia pgAdmin.



*Rys. 12. Logo PostgreSQL*

PostgreSQL został wybrany jako system zarządzania bazą danych w opisywanym w tej pracy projekcie ze względu na jego otwartość i liczbę funkcjonalności oferowanych w darmowym systemie. Ważne jest również podobieństwo do komercyjnych rozwiązań, takich jak bazy Oracle (PL/SQL) – dzięki temu przeniesienie systemu na dużo bardziej wymagającą odmianę nie powinno stanowić zbyt dużego problemu.

1. Opracowanie projektu wielowarstwowej rozproszonej aplikacji internetowej

Sklep z multimediami to idealne miejsce do wykorzystania wielowarstwowej rozproszonej aplikacji internetowej. Tego typu architektura pozwala na obsługę wielu oddziałów sklepu i ciągłe utrzymywanie aktualnego stanu magazynów przy jednoczesnym zabezpieczeniu przed awariami.

* 1. Opis świata rzeczywistego
     1. Opis zasobów ludzkich

Pracownik sklepu może aktualizować bazę produktów poprzez dodawanie lub usuwanie dostępnych egzemplarzy poszczególnych multimediów. Każda pozycja posiada tytuł, cenę, opis, liczbę egzemplarzy i dane magazynu, w którym się znajdują. W zależności od rodzaju multimediów (gra wideo, film, album muzyczny) wyróżniane są dodatkowe informacje takie jak na przykład gatunek, nośnik czy producent. Dodatkowo, każda pozycja jest opisana identyfikatorem ułatwiającym ewidencję posiadanego inwentarza i wyszukiwanie wszystkich dostępnych egzemplarzy.

Klient ma możliwość wyszukiwania produktów, dodawania ich do koszyka i składania zamówienia. Ma również możliwość podejrzenia swoich wszystkich, a także anulowania aktywnych zamówień.

* + 1. Dane techniczne

Klient i pracownik sklepu mają dostęp do sklepu z poziomu przeglądarki internetowej. Zakłada się, że sklep będzie w stanie obsłużyć ponad 1000 użytkowników jednocześnie, a baza danych będzie zawierać kilkadziesiąt tysięcy produktów. Sklep ma kilka oddziałów, więc potrzebna jest integracja danych znajdujących się w więcej niż jednym miejscu. Aplikacja ma być ubezpieczona na ewentualne awarie bazy danych.

* 1. Wymagania funkcjonalne

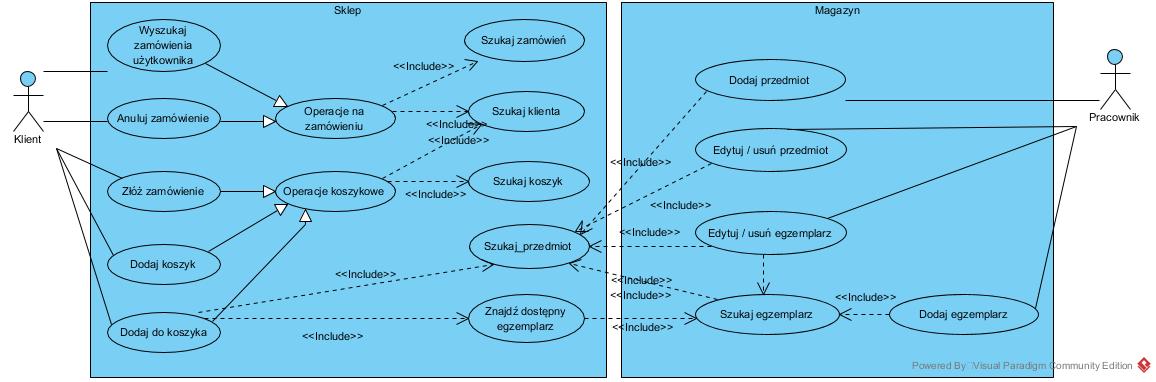
Projektowany system powinien spełniać następujące wymagania funkcjonalne:

* System powinien wspierać przedsiębiorstwo handlowe (w tym przypadku sklep z multimediami)
* Pracownik sklepu może wstawiać nowe przedmioty i ich egzemplarze. Może on zarządzać dostępnym inwentarzem (edycja, usuwanie przedmiotów i egzemplarzy)
* Kupujący tworzy konto, podaje dane osobowe i teleadresowe
* Kupujący może dokonać zakupu dostępnego egzemplarza przy użyciu koszyka i złożenia zamówienia
* Kupujący może zarządzać swoimi aktywnymi zamówieniami (np. je anulować), a także podejrzeć wszystkie dotychczasowe zamówienia.
  1. Wymagania niefunkcjonalne

Projektowany system powinien spełniać następujące wymagania niefunkcjonalne:

* System powinien pracować na popularnych systemach operacyjnych (Windows, MacOS) i w popularnych przeglądarkach internetowych (Google Chrome, Mozilla Firefox, Safari)
* System powinien mieć przejrzysty i czytelny interfejs użytkownika
* System powinien być gotowy na rozszerzenia funkcjonalności (np. możliwość zwrotu zakupionego przedmiotu, czy płatność kartą kredytową)
* System powinien być przygotowany do obsługi co najmniej kilku oddziałów sklepu i utrzymywać w każdym z nich aktualną bazę dostępnych przedmiotów.
  1. Przypadki użycia

Podczas tworzenia modelu wielowarstwowego rozproszonego systemu internetowego do obsługi przedsiębiorstwa handlowego wyszczególnione zostały przypadki użycia przedstawione na rysunku 13.



*Rys. 13. Diagram przypadków użycia.*

W następnych podrozdziałach przedstawione zostały szczegółowe opisy przypadków użycia, szczególnie ważnych dla tematu tej pracy. Będą to przypadki użycia, które zostaną poddane testom wydajnościowym w rozdziale szóstym.

* + 1. Wyszukaj zamówienia użytkownika

Przypadek użycia „Wyszukaj zamówienia użytkownika” odpowiada za wylistowanie na życzenie użytkownika wszystkich jego zamówień (aktywnych i nieaktywnych).

*Tabela 1. Dane PU "Wyszukaj zamówienia użytkownika"*

|  |  |
| --- | --- |
| **Nazwa PU** | Wyszukaj zamówienia użytkownika |
| **Warunki wstępne** | * Uruchomienie aplikacji * Zalogowanie na konto klienta |
| **Warunki końcowe** | * Uzyskanie listy zamówień klienta lub zwracana jest informacja o braku zamówień |
| **Przebieg podstawowy** | 1. Należy podać dane klienta. 2. Realizuje się PU Operacje\_na zamówieniu - sprawdza dane klienta i zwraca listę jego zamówień |
| **Przebieg alternatywny (brak zamówień)** | 1. Należy podać dane klienta. 2. Realizuje się PU Operacje\_na zamówieniu 3. PU Operacje\_na zamówieniu zwraca pustą listę zamówień – albo z powodu braku klienta lub z powodu braku zamówień u znalezionego klienta |

* + 1. Dodaj przedmiot

Przypadek użycia „Dodaj przedmiot” odpowiada za dodanie nowego przedmiotu do bazy danych przedmiotów. Jest on wywoływany przez pracownika sklepu.

*Tabela 2. Dane PU "Dodaj przedmiot"*

|  |  |
| --- | --- |
| **Nazwa PU** | Dodaj przedmiot |
| **Warunki wstępne** | * Uruchomienie aplikacji * Zalogowanie na konto pracownika sklepu |
| **Warunki końcowe** | * Przedmiot zostaje dodany do bazy przedmiotów lub zwracany jest komunikat o istniejącym przedmiocie |
| **Przebieg podstawowy** | 1. Należy podać wymagane dane przedmiotu: nazwę, typ, nośnik, autora, cenę, kod produktu. 2. Należy sprawdzić, czy podany przedmiot istnieje za pomocą wywołania PU „Szukaj przedmiot” przekazując kod produktu. 3. Jeśli przedmiot nie istnieje – należy zapisać dane do bazy danych. |
| **Przebieg alternatywny (przedmiot już istnieje)** | 1. Należy podać wymagane dane przedmiotu: nazwę, typ, nośnik, autora, cenę, kod produktu. 2. Należy sprawdzić, czy podany przedmiot istnieje za pomocą wywołania PU „Szukaj przedmiot” przekazując kod produktu. 3. Jeśli przedmiot istnieje – zwracany jest komunikat o istniejącym przedmiocie. |

* + 1. Edytuj/Usuń egzemplarz

Przypadek użycia „Edytuj/Usuń egzemplarz” odpowiada za edycję lub usunięcie konkretnego egzemplarza przedmiotu z bazy danych. Edycji podlega status egzemplarza, a także jego przypisanie do konkretnego zamówienia. Jest on wywoływany przez pracownika sklepu z poziomu ekranu wyszukiwania egzemplarzy.

*Tabela 3. Dane PU „Edytuj/Usuń egzemplarz”*

|  |  |
| --- | --- |
| **Nazwa PU** | Usuń egzemplarz |
| **Warunki wstępne** | * Uruchomienie aplikacji * Zalogowanie na konto pracownika sklepu |
| **Warunki końcowe** | * Edytowany egzemplarz jest zapisywany w bazie danych * Usuwany egzemplarz jest usuwany z bazy danych * W przypadku braku egzemplarza – zwracany jest komunikat |
| **Przebieg podstawowy**  **(edycja)** | 1. Należy podać kod egzemplarza i edytowane dane: status lub zamówienie. 2. Należy sprawdzić, czy podany egzemplarz istnieje za pomocą wywołania PU „Szukaj egzemplarza” przekazując kod egzemplarza. 3. Jeśli egzemplarz istnieje – należy aktualizować egzemplarz podanymi danymi. 4. Należy zapisać egzemplarz w bazie danych. |
| **Przebieg podstawowy (usuwanie)** | 1. Należy podać kod egzemplarza. 2. Należy sprawdzić, czy podany egzemplarz istnieje za pomocą wywołania PU „Szukaj egzemplarza” przekazując kod egzemplarza. 3. Jeśli egzemplarz istnieje – usunąć egzemplarz z bazy danych. |
| **Przebieg alternatywny (brak egzemplarza)** | 1. Należy podać kod egzemplarza. 2. Należy sprawdzić, czy podany egzemplarz istnieje za pomocą wywołania PU „Szukaj egzemplarza” przekazując kod egzemplarza. 3. Jeśli egzemplarz nie istnieje – zwracany jest stosowny komunikat. |

* + 1. Anuluj zamówienie

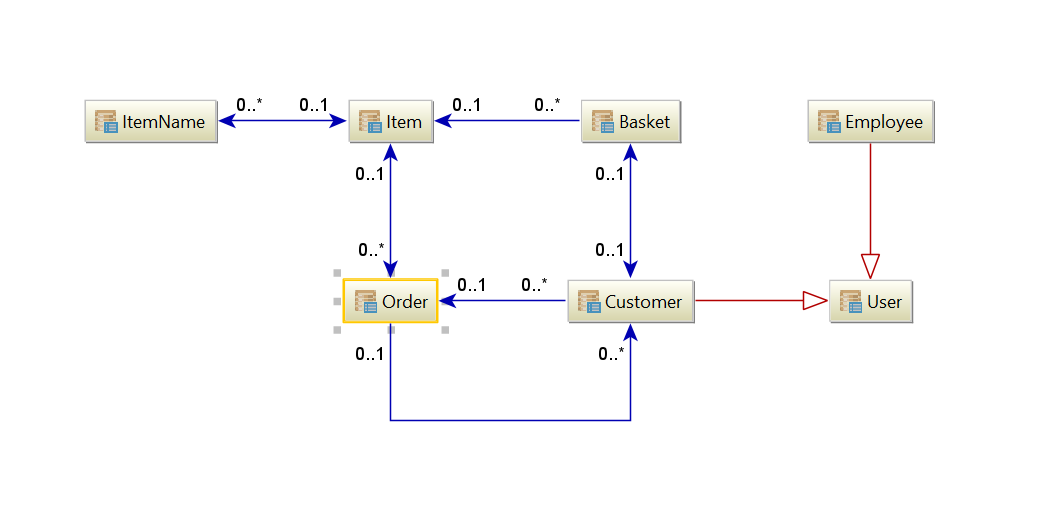
Przypadek użycia „Anuluj zamówienie” odpowiada za anulowanie na życzenie użytkownika konkretnego aktywnego zamówienia spośród wylistowanych w PU „Wyszukaj zamówienia użytkownika”.

*Tabela 4. Dane PU „Anuluj zamówienie”*

|  |  |
| --- | --- |
| **Nazwa PU** | Wyszukaj zamówienia użytkownika |
| **Warunki wstępne** | * Uruchomienie aplikacji * Zalogowanie na konto klienta * Wylistowanie zamówień klienta (PU „Wyszukaj zamówienia użytkownika”) |
| **Warunki końcowe** | * Zmiana statusu zamówienia na „Anulowane” |
| **Przebieg podstawowy** | 1. Należy podać dane klienta i numer zamówienia. 2. Należy sprawdzić, czy podane zamówienie istnieje za pomocą wywołania PU „Operacje na zamówieniu” przekazując numer zamówienia. 3. Jeśli zamówienie istnieje – sprawdzić jego status. 4. Jeśli status zamówienia to „Aktywne” – zmienić status na „Anulowane”. 5. Zmienić status egzemplarzy przypisanych do zamówienia z „Sprzedany” na „Dostępny”. |
| **Przebieg alternatywny (zamówienie nieaktywne)** | 1. Należy podać dane klienta i numer zamówienia. 2. Należy sprawdzić, czy podane zamówienie istnieje za pomocą wywołania PU „Operacje na zamówieniu” przekazując numer zamówienia. 3. Jeśli zamówienie istnieje – sprawdzić jego status. 4. Jeśli status zamówienia jest inny niż „Aktywne” – zwracany jest stosowny komunikat. |
| **Przebieg alternatywny (brak zamówienia)** | 1. Należy podać dane klienta i numer zamówienia. 2. Należy sprawdzić, czy podane zamówienie istnieje za pomocą wywołania PU „Operacje na zamówieniu” przekazując numer zamówienia. 3. Jeśli zamówienie nie istnieje – zwracany jest komunikat. |

* 1. Model danych

Na podstawie zaprojektowanych przypadków użycia i analizy biznesowej utworzony został model danych, z którego korzystać ma projektowany system. Jest to model, który docelowo ma rezydować w bazach danych każdego z oddziałów sklepu.



*Rys. 14. Model danych sklepu z multimediami*

W kolejnych podrozdziałach krótko opisane zostaną poszczególne encje – ich odpowiedzialność i zawartość.

* + 1. ItemName (Nazwa Przedmiotu)

Nazwa Przedmiotu to podstawowy element bazy danych sklepu z multimediami. Posiada najważniejsze informacje danego przedmiotu - takie jak nazwa, kod produktu, cena, czy szczegóły, takie jak typ, data premiery. Zawiera również informację o istniejących egzemplarzach danego przedmiotu.

* + 1. Item (Egzemplarz)

Egzemplarz to uściślenie Nazwy Przedmiotu. To encja opisująca konkretny egzemplarz przedmiotu, posiada swój kod kreskowy, informację o przedmiocie jaki reprezentuje, a także informację o statusie. Status może przyjmować trzy wartości – Dostępny, W koszyku, Sprzedany. Na jego podstawie system generuje wiadomość o dostępności danego przedmiotu.

* + 1. Basket (Koszyk)

Koszyk reprezentuje zestaw egzemplarzy, które użytkownik dodał do koszyka. Każdy użytkownik może posiadać tylko jeden koszyk. Przechowuje on listę egzemplarzy ze statusem „W koszyku”, a także wartość koszyka.

* + 1. Order (Zamówienie)

Zamówienie to kolejny (po koszyku) sposób przetrzymywania informacji o zestawie egzemplarzy. W tym przypadku jednak egzemplarze mają już status „Sprzedane”, a oprócz tego zamówienie posiada informację o swojej wartości, dacie złożenia czy użytkowniku.

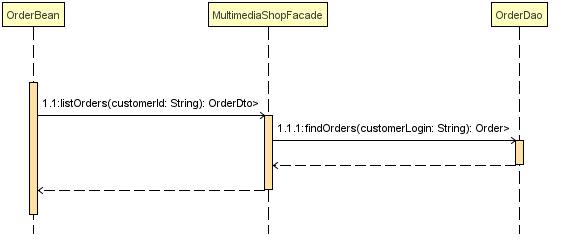
* + 1. User (Użytkownik) i encje powiązane

Użytkownik przechowuje informacje teleadresowe takie jak adres, imię czy nazwisko, a także dane logowania – login i hasło. Wyróżnione zostały dwa typy użytkowników – Klient i Pracownik. Klient dodatkowo posiada informację o złożonych przez niego zamówieniach, a także o koszyku. Z kolei Pracownik dysponuje identyfikatorem pracownika, który ułatwia identyfikację i przyspiesza przydzielanie dostępów.

1. Szczegóły projektowe
   1. Diagramy sekwencji

W tym podrozdziale zostaną przedstawione diagramy sekwencji powiązane z przypadkami użycia opisanymi w rozdziale poprzednim.

* + 1. Wyszukaj zamówienia użytkownika



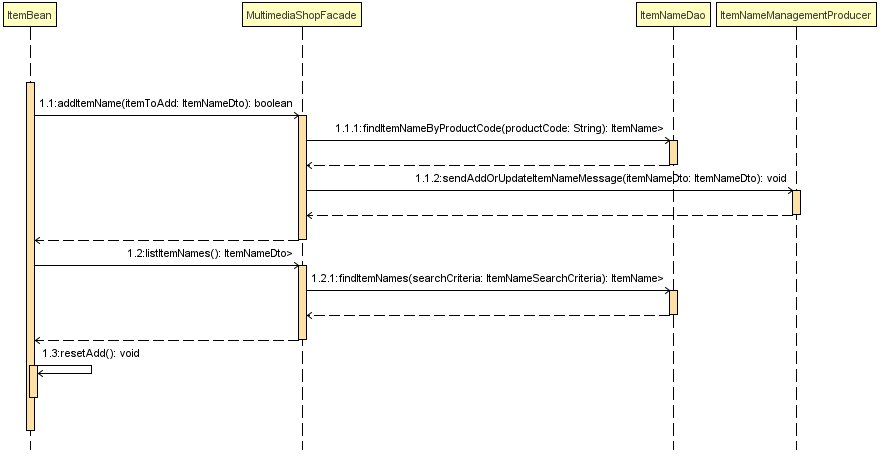
*Rys. 15. Diagram sekwencji "Wyszukaj zamówienia użytkownika"*

Dzięki temu, że każdy z oddziałów sklepu posiada własną bazę danych, która jest ciągle synchronizowana z pozostałymi oddziałami, przypadek użycia „Wyszukaj zamówienia użytkownika” staje się nieskomplikowany. Wystarczające jest tutaj odwołanie do fasady i system w prosty sposób wyciąga listę zamówień ze swojej bazy.

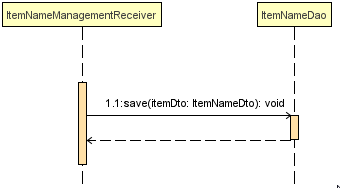
Przypadek użycia definiuje jako wymaganie – sprawdzenie, czy istnieje konkretny użytkownik. To wymaganie jest spełnione poprzez zalogowanie się użytkownika do systemu, dlatego też nie ma tutaj konieczności ponownego sprawdzania.

* + 1. Dodaj przedmiot

Diagram sekwencji dodawania przedmiotu (jak i każdy kolejny korzystający z wiadomości JMS) można podzielić na dwie części – część prowadzącą do wysłania wiadomości i część po odebraniu wiadomości JMS.



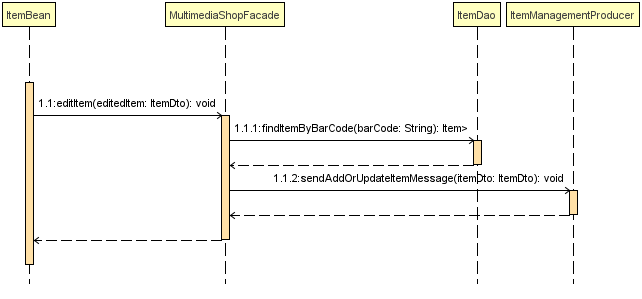
*Rys. 16. Diagram sekwencji PU "Dodaj przedmiot" - część przed wysłaniem wiadomości JMS*



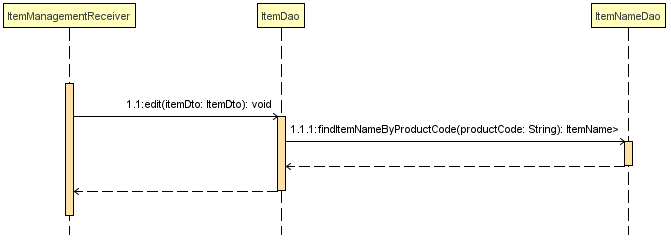
*Rys. 17. Diagram sekwencji PU "Dodaj przedmiot" - część po odebraniu wiadomości JMS*

* + 1. Edytuj egzemplarz

W przypadku edycji istniejącego egzemplarza do fasady zostaje wysłane zapytanie z zawartością tego egzemplarza. Następnie sprawdzane jest, czy taki egzemplarz już istnieje – jeśli tak, wysyłana jest wiadomość o edycji. Po odebraniu wiadomości odbywa się dodatkowe sprawdzenie, czy egzemplarz ma poprawnie przypisaną nazwę przedmiotu, a następnie dane są zapisywane do bazy danych.



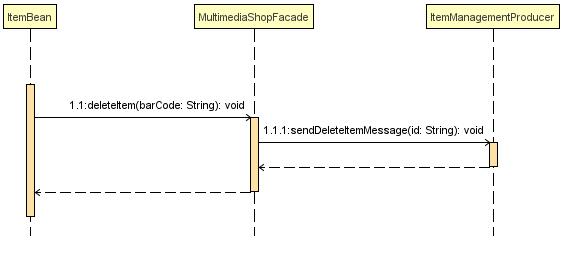
*Rys. 18. Diagram sekwencji PU "Edytuj egzemplarz" - część przed wysłaniem wiadomości JMS*



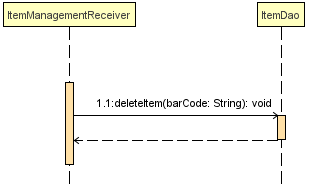
*Rys. 19. Diagram sekwencji PU "Edytuj egzemplarz" - część po wysłaniu wiadomości JMS*

* + 1. Usuń egzemplarz

W przypadku próby usunięcia egzemplarza wiadomość jest wysyłana od razu po otrzymaniu zapytania przez fasadę. Wiadomość jest odbierana przez komponent nasłuchujący, a następnie egzemplarz jest usuwany z bazy danych. Dzięki użyciu adnotacji Hibernate programista nie musi samodzielnie obsługiwać ewentualnych zależności – odpowiadają za to Hibernate i JPA.



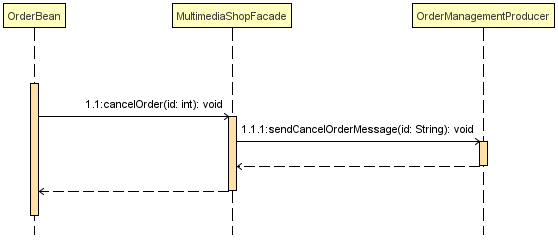
*Rys. 20. Diagram sekwencji PU "Usuń egzemplarz" - część przed wysłaniem wiadomości JMS*



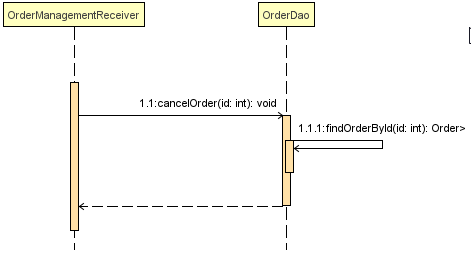
*Rys. 21. Diagram sekwencji PU "Usuń egzemplarz" - część po odebraniu wiadomości JMS*

* + 1. Anuluj zamówienie

Przypadek użycia „Anuluj zamówienie” różni się od „Usuń egzemplarz” tym, że docelowo zamówienie nie ma być usunięte z bazy danych, a ma się zmienić jego status. Dlatego też po otrzymaniu wiadomości żądającej anulowania zamówienia system wyszukuje odpowiednie zamówienie w bazie danych i zmienia jego status na „Anulowane”. Zmieniany jest również status wszystkich egzemplarzy przypisanych do zamówienia z „Sprzedany” na „Dostępny”.



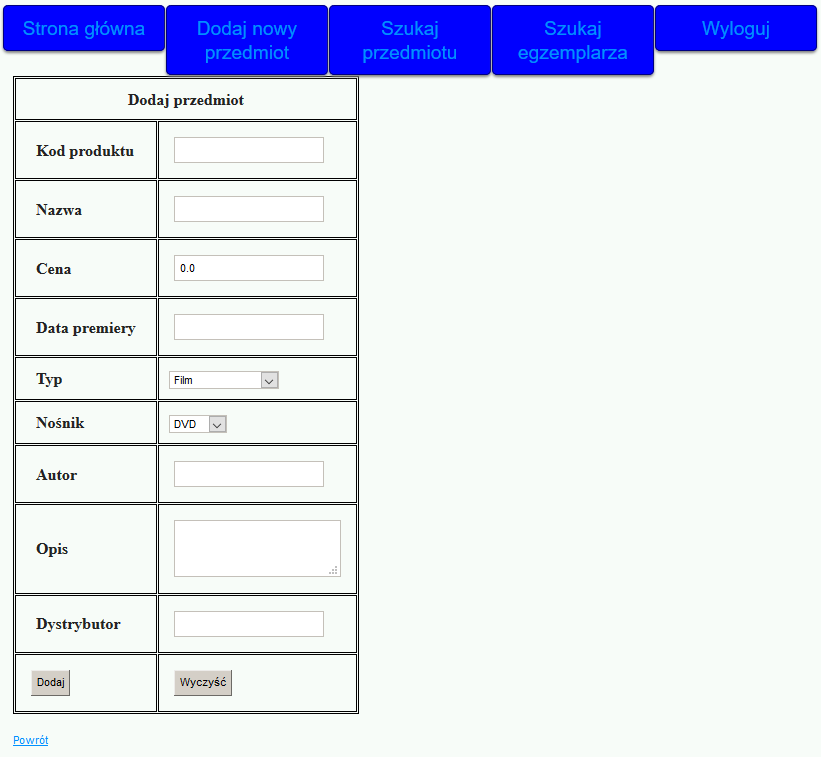
*Rys. 22. Diagram sekwencji PU "Anuluj zamówienie" - część przed wysłaniem wiadomości JMS*



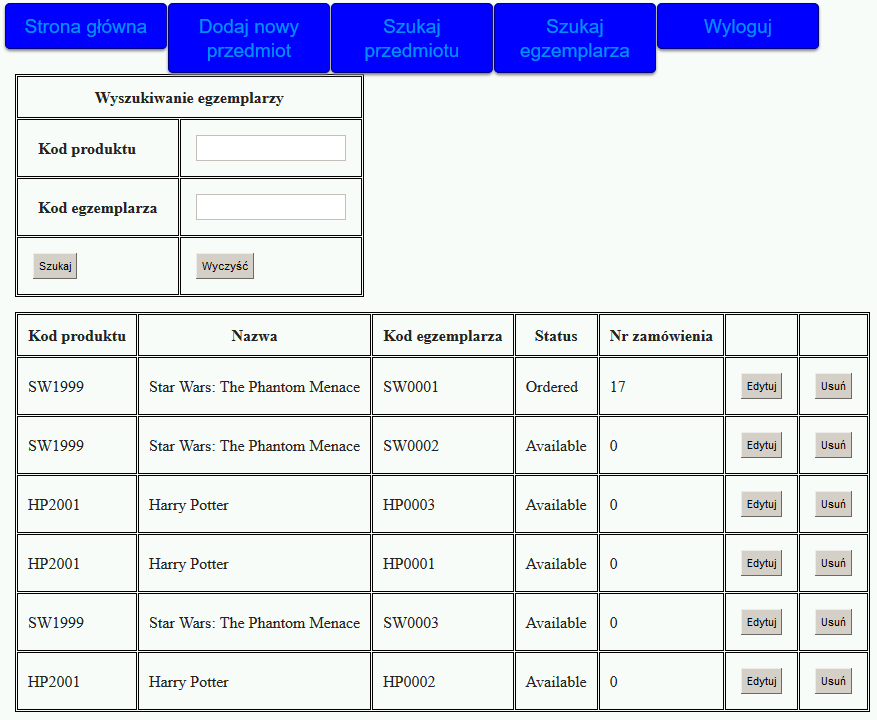
*Rys. 23. Diagram sekwencji PU "Anuluj zamówienie" - część po odebraniu wiadomości JMS*

* 1. Interfejs użytkownika

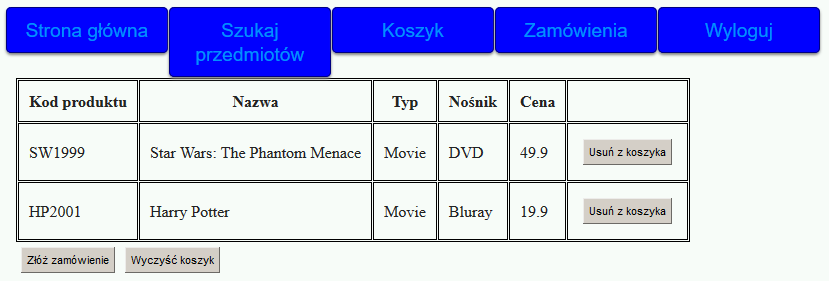
Głównymi założeniami interfejsu użytkownika są przejrzystość i rozszerzalność. Przygotowane szkice interfejsu to tak naprawdę szkielety, które w nieskomplikowany sposób można rozszerzać, a także modyfikować ich stronę wizualną za pomocą arkuszy styli CSS. Przedstawiony poniżej interfejs jest najprostszą wersją (bez użycia CSS) – wersją, którą każde przedsiębiorstwo może dostosowywać do własnych upodobań.



*Rys. 24. Ekran dodawania przedmiotu przez pracownika*



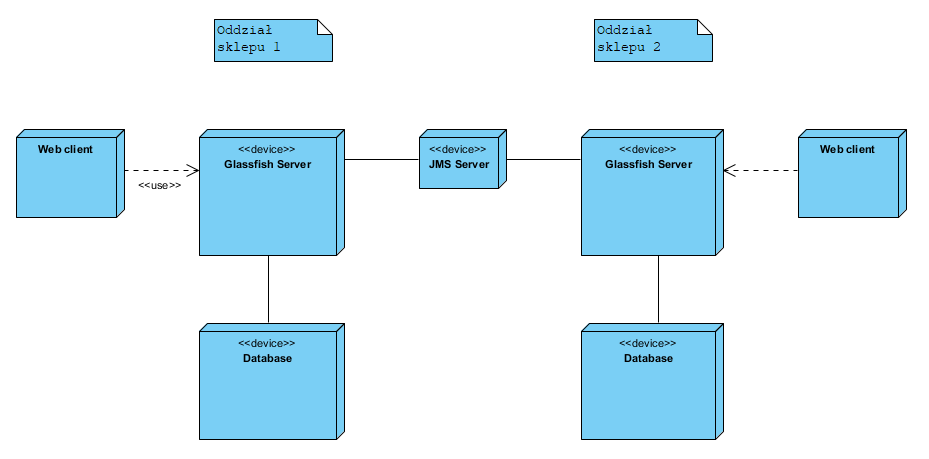
*Rys. 25. Ekran wyszukiwania egzemplarzy przez pracownika.*



*Rys. 26. Ekran koszyka widziany z poziomu klienta.*

1. Szczegóły implementacyjne
   1. Architektura systemu

Wielowarstwowa rozproszona aplikacja internetowa do obsługi przedsiębiorstwa handlowego została zaprojektowana w taki sposób, aby każdy z oddziałów sklepu posiadał swoją własną bazę danych zawierającą informacje o ogólnej dostępności przedmiotów, użytkownikach czy zamówieniach. Każde działanie wykonywane na lokalnej bazie danych powinno być propagowane do pozostałych oddziałów za pomocą technologii JMS.



*Rys. 27. Topologia systemu.*

System składa się z serwera JMS i szeregu serwerów aplikacji (każdy posiada swoją bazę danych). Serwery komunikują się ze sobą przy użyciu wątku JMS, tak że każda zmiana w bazie danych powoduje wysłanie wiadomości do wszystkich subskrybujących. W kolejnym podrozdziale zostaną opisane szczegóły tej komunikacji.

* 1. Rozproszoność systemu

Zaprojektowana aplikacja korzysta z szeregu wątków – każdy z nich jest przystosowany do obsługi poszczególnych typów informacji w bazie danych (Przedmiot, Zamówienie, Użytkownik). Wiadomość jest wysyłana przez producenta (producer), czyli specjalny bezstanowy komponent typu EJB. Producent wysyła wiadomość na konkretny wątek, wiadomość zawiera obiekt, który użytkownik chce dodać lub edytować w bazie danych, lub identyfikator obiektu do usunięcia z bazy danych. Nastepnie wiadomość jest przekazywana do serwera JMS, który rozsyła ją do wszystkich subskrybentów.

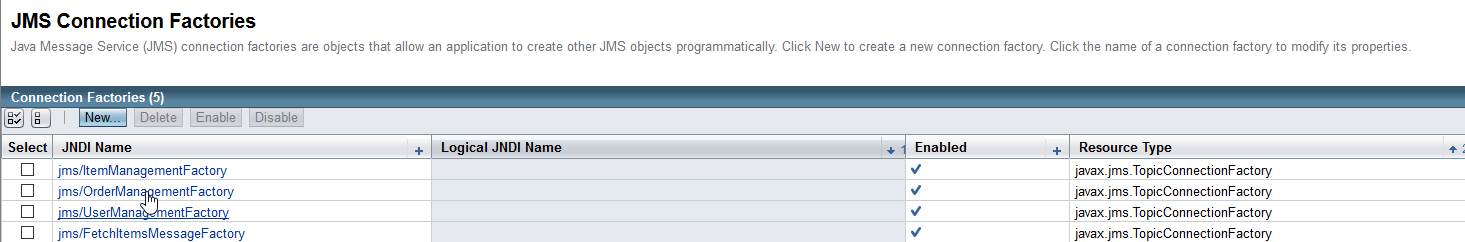
*Listing. 1. Przykładowy producent wiadomości JMS*

@Stateless  
public class ItemManagementProducer {  
 @Resource(lookup = "jms/ItemManagementFactory")  
 private ConnectionFactory connectionFactory;  
  
 @Resource (lookup = "jms/ItemManagement")  
 private Topic topic;  
  
 public void sendAddOrUpdateItemMessage(ItemDto itemDto) {  
 try {  
 Connection connection = connectionFactory.createConnection();  
 Session session = connection.createSession(false, Session.CLIENT\_ACKNOWLEDGE);  
 MessageProducer messageProducer = session.createProducer(topic);  
  
 ObjectMessage message = session.createObjectMessage();  
  
 message.setObject(itemDto);  
 messageProducer.send(message);  
 messageProducer.close();  
 connection.close();  
  
 } catch (JMSException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
  
 public void sendDeleteItemMessage(String id) {  
 try {  
 Connection connection = connectionFactory.createConnection();  
 Session session = connection.createSession(false, Session.CLIENT\_ACKNOWLEDGE);  
 MessageProducer messageProducer = session.createProducer(topic);  
  
 TextMessage message = session.createTextMessage();  
  
 message.setText(id);  
 messageProducer.send(message);  
 messageProducer.close();  
 connection.close();  
  
 } catch (JMSException ex) {  
 ex.printStackTrace();  
 }  
 }  
}

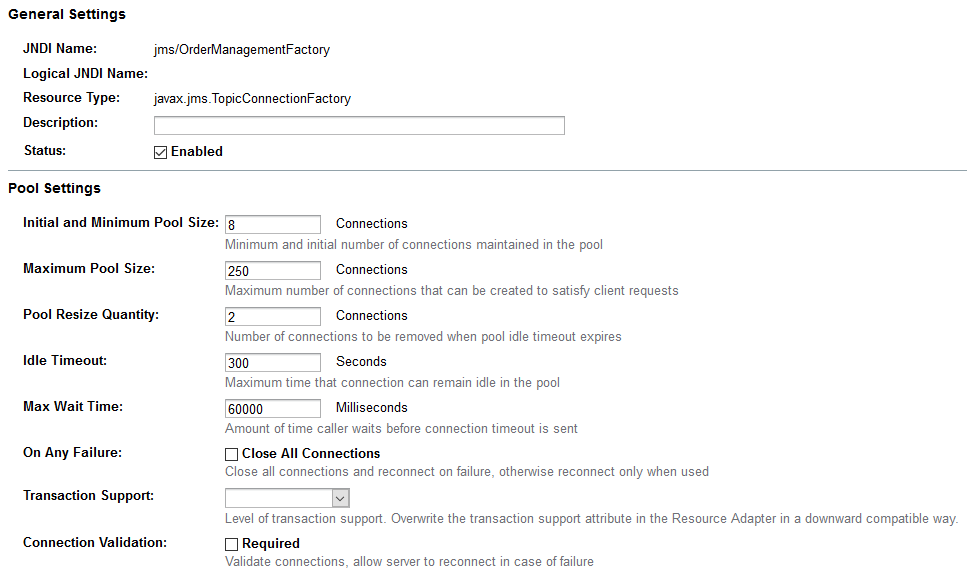
Każdy subskrybent posiada komponenty typu MDB (message-driven bean), które są w stanie w sposób asynchroniczny obsłużyć otrzymane żądanie. Komponenty te rozszerzają interfejs MessageListener. Odpowiadają za wyłuskanie z otrzymanej wiadomości danych do przetworzenia, a następnie przekazania ich do warstwy integracji i, w konsekwencji, do bazy danych danego oddziału sklepu.

*Listing. 2. Przykładowy komponent typu MDB nasłuchujący wiadomości JMS*

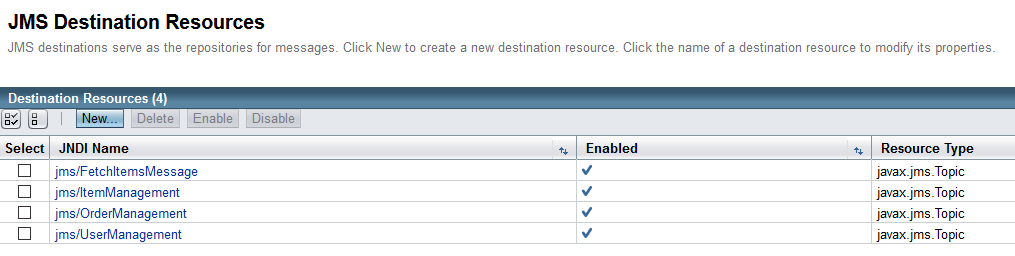
@MessageDriven(activationConfig = {  
 @ActivationConfigProperty(propertyName = "clientId", propertyValue = "jms/ItemManagement"),  
 @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "jms/ItemManagement"),  
 @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable"),  
 @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "jms/ItemManagement"),  
 @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic")  
})  
public class ItemManagementReceiver implements MessageListener {  
 @Resource  
 private MessageDrivenContext mdc;  
  
 @Inject  
 private ItemDao dao;  
  
 public ItemManagementReceiver() {  
 }  
  
 @Override  
 public void onMessage(Message message) {  
 try {  
 if (message instanceof ObjectMessage) {  
 ObjectMessage msg = (ObjectMessage) message;  
 ItemDto item = (ItemDto) msg.getObject();  
 if (item != null && item.isEdited()) {  
 dao.edit(item);  
 } else if (item != null) {  
 dao.save(item);  
 }  
 message.acknowledge();  
 } else if (message instanceof TextMessage) {  
 TextMessage msg = (TextMessage) message;  
 String id = msg.getText();  
 if (id != null)  
 dao.deleteItem(id);  
 message.acknowledge();  
 }  
 } catch (JMSException e) {  
 e.printStackTrace();  
 mdc.setRollbackOnly();  
 } catch (Throwable te) {  
 te.printStackTrace();  
 }  
 }  
}



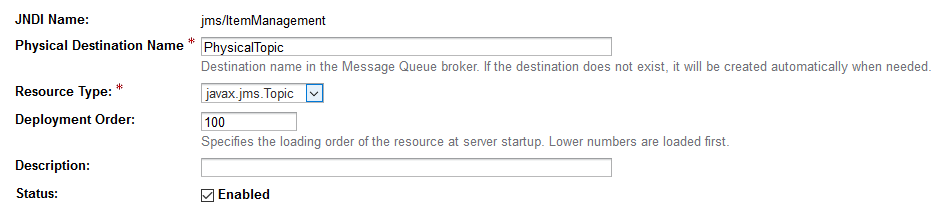
*Rys. 28. Konfiguracja fabryk połączeń JMS*



*Rys. 29. Przykładowa konfiguracja fabryki połączeń.*



Rys. 30. Konfiguracja wątków JMS



*Rys. 31. Konfiguracja przykładowego wątku JMS.*

* 1. Bezpieczeństwo

W zrealizowanym projekcie wielowarstwowej rozproszonej aplikacji internetowej nie był kładziony nacisk na bezpieczeństwo, wynikający z uwierzytelniania czy przechowywania danych użytkowników. Skupiono się na zapewnieniu, że w przypadku nagłej awarii w jednym z oddziałów przedsiębiorstwa handlowego system nadal będzie funkcjonował. Ten poziom bezpieczeństwa w głównej mierze zapewnia użycie technologii JMS i synchronizacji lokalnych baz danych. Gdy dojdzie do awarii w jednym z oddziałów sklepu, pozostałe oddziały nadal są w stanie się ze sobą komunikować i wymieniać informacjami. Pozwala to też na ewentualne rozszerzenie systemu o kolejne oddziały, bo nie są one od siebie zależne. Gdy system danego oddziału zostanie naprawiony – wszystkie wiadomości, które w międzyczasie zostały rozesłane nadal do niego trafią. Jest to zasługa użycia trwałej subskrypcji (durable).

* + 1. Trwała subskrypcja (durable subsription)

Wiadomość jest wysyłana do subskrybenta do czasu, aż ten poinformuje serwer o jej otrzymaniu. Odpowiada za to właściwość Acknowledgment (potwierdzenie). Informacja potwierdzająca może być wysyłana automatycznie na koniec przetwarzania, lub, jeśli użytkownik tego wymaga, manualnie w miejscu wyznaczonym przez programistę. W przypadku przedstawianego tu systemu użyte zostały explicite potwierdzenia przetworzenia wiadomości. Dzięki temu, jeśli pojawi się jakiś problem podczas przetwarzania, lub wystąpi awaria systemu – wiadomość nie przepadnie i zostanie przetworzona po ustabilizowaniu się sytuacji w oddziale sklepu. Trwała subskrypcja zwiększa niezawodność systemu, ale kosztem zmniejszenia wydajności.

*Listing. 3. Użycie potwierdzenia przetworzenia wiadomości w trwałej subskrypcji*

@Override  
public void onMessage(Message message) {  
 try {  
 if (message instanceof ObjectMessage) {  
 ObjectMessage msg = (ObjectMessage) message;  
 UserDto user = (UserDto) msg.getObject();  
 LoggingSupport.logTimeToConsole("OrderManagementReceiver: Received message with user " + user.getLogin());  
 if (user != null) {  
 dao.save(user);  
 }  
 message.acknowledge();  
 }  
 } catch (JMSException e) {  
 e.printStackTrace();  
 mdc.setRollbackOnly();  
 } catch (Throwable te) {  
 te.printStackTrace();}

* 1. Interfejs użytkownika

Interfejs użytkownika został zrealizowany z języku XHTML wspieranym przez JavaServer Faces. W łatwym rozplanowaniu interfejsu pomagały takie funkcjonalności JSF, jak tablice danych (DataTable) czy możliwość zarządzania wyświetlaniem elementów (parametr „rendered”).

*Listing. 4. Kod przykładowej strony (w tym przypadku – wyświetlanie zamówień użytkownika)*

<ui:composition template="template/commonLayout.xhtml">  
 <ui:define name="content">  
 <div class="content">  
 <h:form>  
 <f:facet name="header">  
 <h:outputText value="Twój koszyk" />  
 </f:facet>  
 <h:dataTable value="#{orderBean.listOrders(customerLoginBean.login)}" var="order" columns="11" border="1" cellpadding="10" cellspacing="1">  
 <h:column><f:facet name="header">Identyfikator zamówienia</f:facet>#{order.id}</h:column>  
 <h:column><f:facet name="header">Wartość</f:facet>#{order.value}</h:column>  
 <h:column><f:facet name="header">Status</f:facet>#{order.status}</h:column>  
 <h:column><f:facet name="header">Data złożenia zamówienia</f:facet>#{order.placementDate}</h:column>  
 <h:column>  
 <h:commandButton value="Szczegóły" action="#{orderBean.orderDetails(order)}" />  
 </h:column>  
 <h:column>  
 <h:commandButton value="Anuluj zamówienie" action="#{orderBean.cancelOrder(order)}" />  
 </h:column>  
 </h:dataTable>  
 </h:form>  
  
 <h:form rendered="#{orderBean.detailsEnabled}">  
 <h:panelGrid id="orderDetails" columns="2" border="1" cellpadding="10" cellspacing="1" >  
 <f:facet name="header">  
 <h3>Szczegóły zamówienia #{orderBean.details.id}</h3>  
 </f:facet>  
 <h:outputLabel value="Identyfikator zamówienia" style="font-weight:bold"/>  
 <h:outputText value="#{orderBean.details.id}"/>  
 <h:outputLabel value="Status" style="font-weight:bold"/>  
 <h:outputText value="#{orderBean.details.status}"/>  
 <h:outputLabel value="Wartość" style="font-weight:bold"/>  
 <h:outputText value="#{orderBean.details.value}"/>  
 <h:outputLabel value="Data zamówienia" style="font-weight:bold"/>  
 <h:outputText value="#{orderBean.details.placementDate}"/>  
  
 <h:dataTable value="#{orderBean.details.items}" var="item" columns="11" border="1" cellpadding="10" cellspacing="1">  
 <h:column><f:facet name="header">Kod produktu</f:facet>#{item.itemNameDto.productCode}</h:column>  
 <h:column><f:facet name="header">Nazwa</f:facet>#{item.itemNameDto.name}</h:column>  
 <h:column><f:facet name="header">Typ</f:facet>#{item.itemNameDto.mediaType}</h:column>  
 <h:column><f:facet name="header">Cena</f:facet>#{item.itemNameDto.price}</h:column>  
 </h:dataTable>  
  
 <h:commandButton value="Anuluj zamówienie" action="#{orderBean.cancelOrder(orderBean.details)}" />  
 <h:commandButton value="Zamknij szczegóły" action="#{orderBean.disableDetails}" />  
 </h:panelGrid>  
 </h:form>  
 <p>  
 <a href="customerIndex.xhtml">Powrót</a>  
 </p>  
 </div>  
 </ui:define>  
</ui:composition>

Kod napisany w oparciu o JSF pozwala na używanie wielu technologii i zapewnia przy tym „przenośność” klienta. Raz napisany kod w JSF może być wykorzystywany na różnego typu urządzeniach odbiorczych i w różnego rodzaju technologiach.

1. Pomiary wydajności i analiza wyników

Zaprojektowany i zaimplementowany system został poddany testom wydajnościowym. Do testów został wybrany przekrój funkcjonalności, w celu zapewnienia jak najdokładniejszego obrazu wydajności. W poprzednich rozdziałach opisane zostały przypadki użycia, na których testach skupiono się w tej pracy. Są to:

* Wyszukiwanie zamówień użytkownika
* Dodanie nowego przedmiotu
* Edycja egzemplarza

Każdy z przypadków użycia został przebadany za pomocą metryk JMeter, a także za pomocą pomiarów czasu. Dzięki takiemu podejściu możliwe było sprawdzenie, jak system radzi sobie z obciążeniem, jak efektywny jest JMS, a także jak szybko system wykonuje polecenia użytkownika.

* 1. JMeter

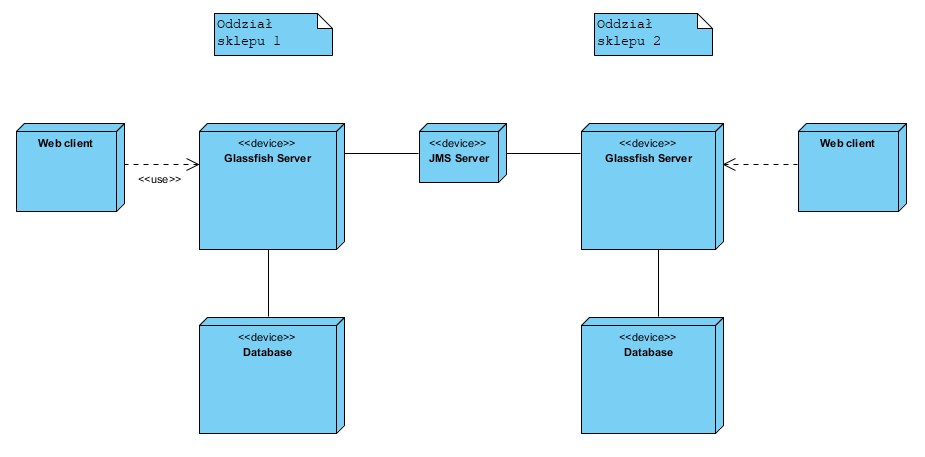
Apache JMeter to otwartoźródłowa aplikacja zaprojektowana do przeprowadzania testów wydajnościowych i funkcyjnych. JMeter pozwala na testowanie takich technologii jak:

* Technologie sieciowe (HTTP, Java, NodeJS itp.)
* Serwisy sieciowe (SOAP, REST)
* Bazy danych (za pomocą JDBC)
* JMS
* Obiekty Java
* Inne (TCP, poczta, FTP).

Lista obsługiwanych funkcjonalności sprawia, że jest to dobre rozwiązanie do testowania wielowarstwowych rozproszonych aplikacji internetowych, takich jak opisywana w tej pracy aplikacja do obsługi sklepu z multimediami. Co ważne, JMeter pozwala na szybkie przygotowywanie Test Planów, a także potrafi wczytywać dane z najpopularniejszych formatów (HTML, JSON, XML), dzięki czemu dobrze współpracuje z komponentami sieciowymi JSF[9].

* 1. Środowiska testowe

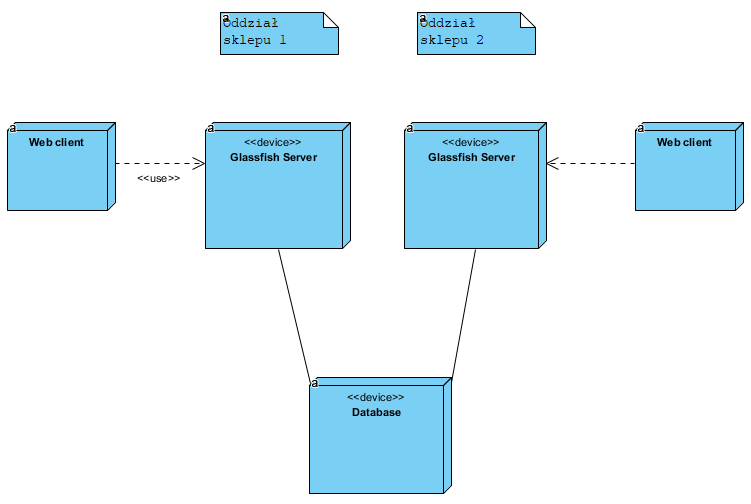
Do przeprowadzenia testów użyto dwóch konfiguracji – z JMS i bez JMS. użycia Konfiguracja pierwsza to konfiguracja opisywana we wcześniejszej pracy. Jej schemat przedstawia się następująco:



*Rys. 32. Konfiguracja testowa 1.*

W tym przypadku zbadane zostały czasy wykonania przypadku użycia. Testy przeprowadzono dla wspomnianych wcześniej przypadków.

Druga konfiguracja testowa, to konfiguracja nie korzystająca z synchronizacji baz danych za pomocą JMS. W tej konfiguracji baza danych jest jedna scentralizowana, a wszystkie oddziały sklepu odwołują się do niej bezpośrednio.



*Rys. 33. Konfiguracja testowa 2.*

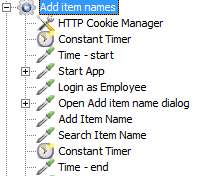
Testy zostały przeprowadzone dla następujących przypadków:

* Dodanie przedmiotu
* Edycja egzemplarza
* Wyszukiwanie zamówień użytkownika
* Usunięcie egzemplarza

W kolejnych podrozdziałach zostaną omówione poszczególne konfiguracje.

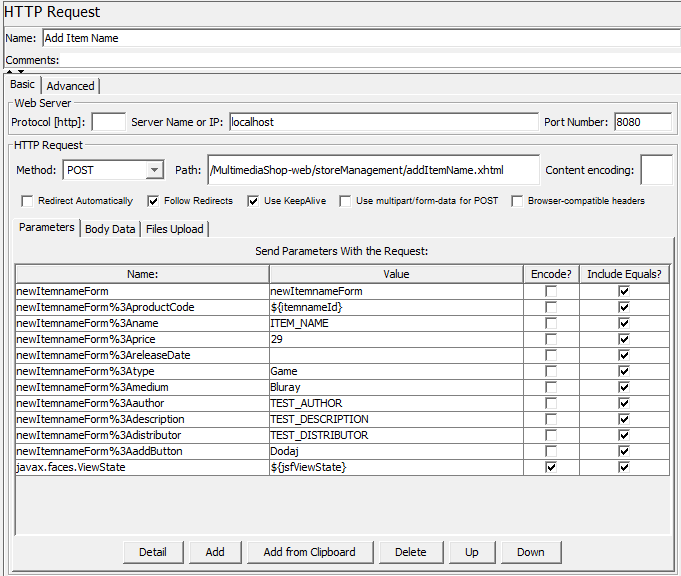
* + 1. Dodanie przedmiotu

Konfiguracja JMeter dla tego przypadku przedstawia się następująco:



*Rys. 34. Konfiguracja JMeter dla dodawania przedmiotów*

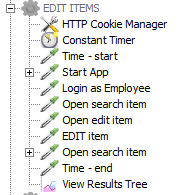
Test rozpoczyna się od zalogowania pracownika na konto. Następnie pracownik przechodzi do okna dodawania przedmiotu, uzupełnia dane i wysyła je na serwer. Test skonfigurowany jest dla dziesięciu jednoczesnych połączeń i korzysta z wiadomości http typu POST.



*Rys. 35. Dodaj przedmiot - Żądanie HTTP*

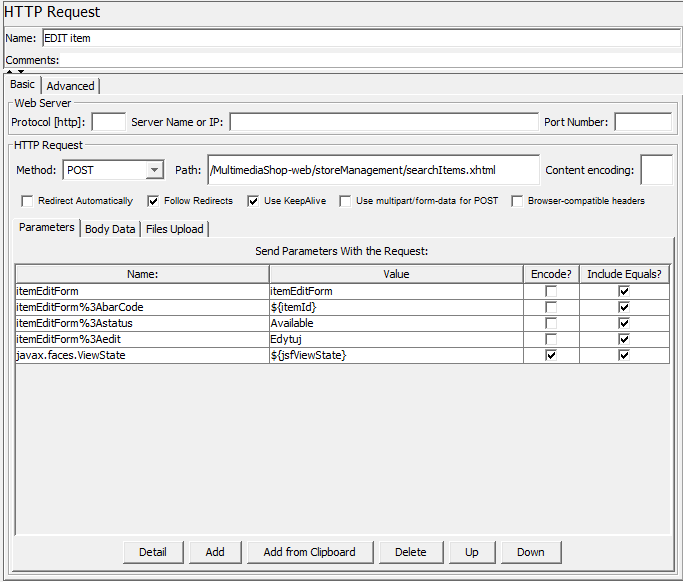
Przed zakończeniem testu pracownik sprawdza w oknie „Wyszukaj przedmiot” czy przedmiot faktycznie został dodany. Komponenty „Time – start” i „Time – end” służą do mierzenia czasu wykonywania przypadku użycia. To na podstawie ich wskazań wykonana została analiza wyników przedstawiona w kolejnych podrozdziałach.

* + 1. Edycja egzemplarza



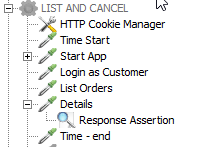
*Rys. 36. Konfiguracja JMeter dla edycji egzemplarzy*

Test rozpoczyna się od zalogowania pracownika na konto. Następnie pracownik przechodzi do okna wyszukiwania egzemplarza i znajduje konkretny egzemplarz. Wybiera „Edytuj”, uzupełnia dane i wysyła na serwer. Następnie sprawdzane jest, czy egzemplarz faktycznie został wyedytowany.



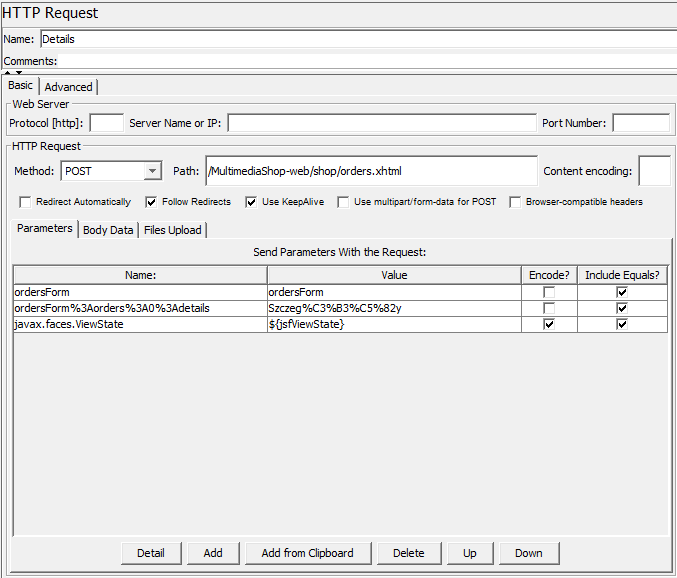
*Rys. 37. Edytuj egzemplarz - Żądanie HTTP*

* + 1. Wyszukaj zamówienia użytkownika



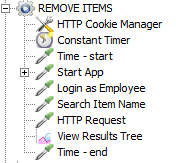
*Rys. 38. Konfiguracja JMeter dla wyszukiwania zamówień*

Test rozpoczyna się od zalogowania klienta na konto. Następnie klient przechodzi do okna wyszukiwania zamówień. Wyświetlona zostaje lista wszystkich zamówień zalogowanego użytkownika.



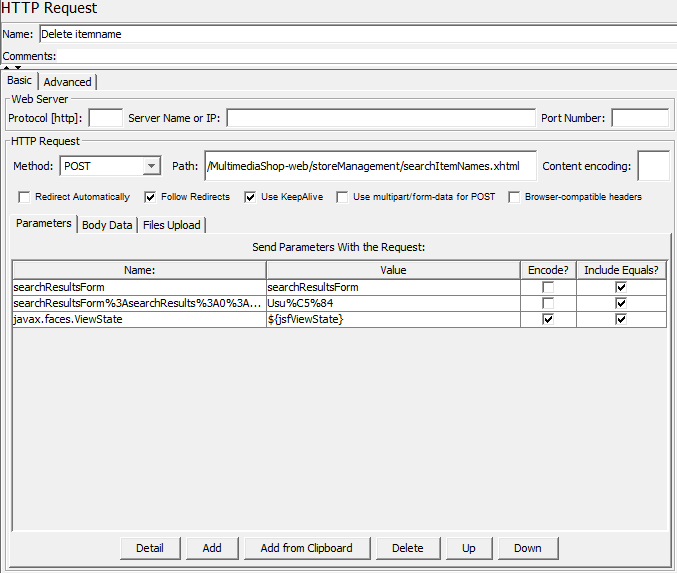
*Rys. 39. Wyszukaj zamówienia - Żądanie http*

* + 1. Usuń przedmiot



*Rys. 40. Konfiguracja JMeter dla usuwania przedmiotów*

Test rozpoczyna się od zalogowania pracownika na konto. Następnie pracownik przechodzi do okna wyszukiwania przedmiotu i znajduje konkretny przedmiot. Wybiera „Usuń”. Następnie sprawdzane jest, czy przedmiot faktycznie został usunięty (wraz z jego egzemplarzami).



*Rys. 41. Usuń przedmiot - Żądanie HTTP*

* 1. Otrzymane wyniki

Dla każdego z przypadków zostało wykonane 20 prób. Poniżej uzyskane wyniki i tabele zbiorcze z danymi liczbowymi. Czas w osi pionowej podawany jest w milisekundach.

W poniższych tabelach czas podawany jest w milisekundach.

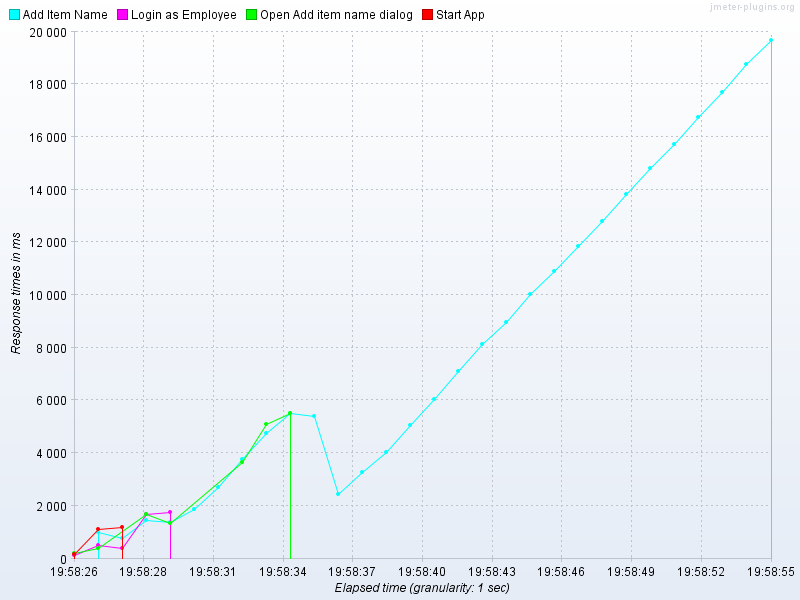
*Tabela 5. Wyniki testów w konfiguracji używającej JMS*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr próby | DODAWANIE PRZEDMIOTU | EDYCJA EGZEMPLARZA | WYSZUKIWANIE ZAMÓWIEŃ | USUWANIE EGZEMPLARZA |
| 1 | 4526 | 2609 | 65 | 1618 |
| 2 | 4728 | 2515 | 62 | 1572 |
| 3 | 4627 | 2540 | 41 | 1554 |
| 4 | 4827 | 2521 | 303 | 1546 |
| 5 | 4926 | 2530 | 121 | 1557 |
| 6 | 4473 | 2536 | 226 | 1575 |
| 7 | 4294 | 2508 | 55 | 1556 |
| 8 | 4633 | 2503 | 50 | 1555 |
| 9 | 4451 | 2546 | 37 | 1548 |
| 10 | 4290 | 2527 | 43 | 1550 |
| 11 | 4745 | 2542 | 77 | 1677 |
| 12 | 5215 | 2638 | 38 | 1582 |
| 13 | 5309 | 2523 | 33 | 1568 |
| 14 | 4915 | 2522 | 46 | 1571 |
| 15 | 5016 | 2517 | 33 | 1558 |
| 16 | 4827 | 2547 | 78 | 1559 |
| 17 | 4727 | 2524 | 31 | 1561 |
| 18 | 4950 | 2507 | 45 | 1555 |
| 19 | 5140 | 2515 | 33 | 1557 |
| 20 | 4819 | 2546 | 45 | 1558 |

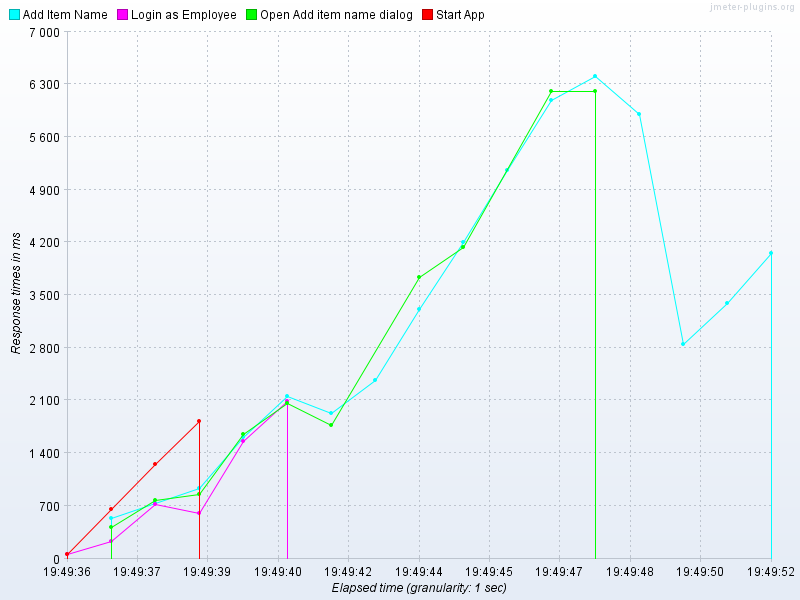
*Tabela 6. Wyniki testów przeprowadzonych z konfiguracji z centralną bazą danych.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr próby | DODAWANIE PRZEDMIOTU | EDYCJA EGZEMPLARZA | WYSZUKIWANIE ZAMÓWIEŃ | USUWANIE EGZEMPLARZA |
| 1 | 3666 | 2516 | 154 | 1594 |
| 2 | 3766 | 2457 | 129 | 1537 |
| 3 | 3867 | 2459 | 138 | 1533 |
| 4 | 3656 | 2459 | 162 | 1532 |
| 5 | 3651 | 2453 | 151 | 1533 |
| 6 | 3654 | 2466 | 206 | 1553 |
| 7 | 3656 | 2453 | 192 | 1531 |
| 8 | 3655 | 2454 | 190 | 1531 |
| 9 | 3655 | 2455 | 231 | 1535 |
| 10 | 3660 | 2456 | 197 | 1529 |
| 11 | 3783 | 2526 | 187 | 1595 |
| 12 | 3683 | 2454 | 198 | 1534 |
| 13 | 3883 | 2453 | 202 | 1534 |
| 14 | 3656 | 2455 | 217 | 1534 |
| 15 | 3658 | 2452 | 231 | 1532 |
| 16 | 3656 | 2455 | 285 | 1533 |
| 17 | 3655 | 2453 | 299 | 1530 |
| 18 | 3663 | 2455 | 330 | 1531 |
| 19 | 3667 | 2452 | 335 | 1530 |
| 20 | 3660 | 2451 | 292 | 1534 |

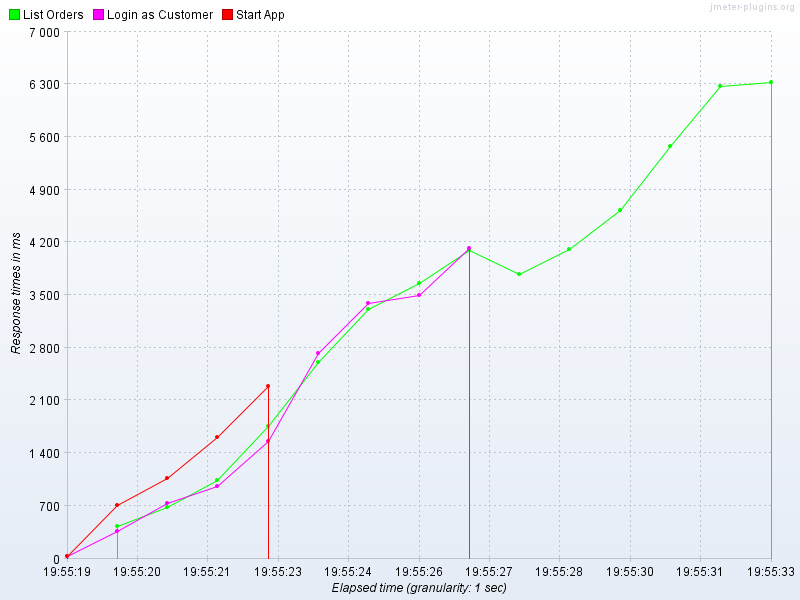
Przeprowadzono również test obciążeniowy na przykładzie tworzenia nowych przedmiotów i wyszukiwania zamówień użytkowników – dane operacje wykonano na 400 wątkach jednocześnie.



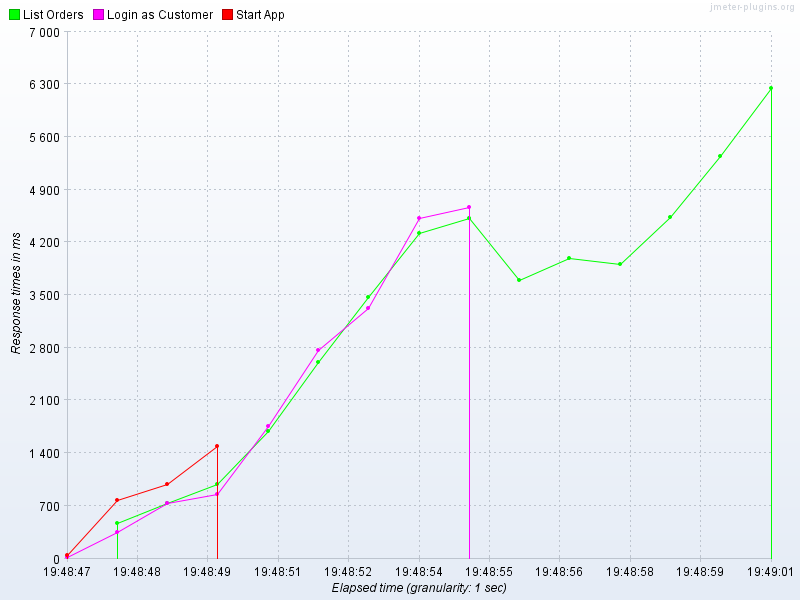
*Rys. 42. Wyniki testu obciążeniowego dla przypadku Dodaj Przedmiot na konfiguracji z centralną użyciem JMS*



*Rys. 43. Wyniki testu obciążeniowego dla przypadku Dodaj Przedmiot na konfiguracji z centralną bazą danych*



*Rys. 44. Wyniki testu obciążeniowego dla przypadku Wyszukaj Zamówienia na konfiguracji z użyciem JMS*



*Rys. 45. Wyniki testu obciążeniowego dla przypadku Wyszukaj Zamówienia na konfiguracji z centralną bazą danych*

* 1. Analiza wyników

Zgodnie z oczekiwaniami konfiguracja używająca JMS okazała się być minimalnie wolniejsza od konfiguracji z centralną bazą danych. Otrzymane wyniki dotyczą jednak testów w dość hermetycznym środowisku, gdzie odległości między serwerami są bardzo małe, a same serwery znajdują się w jednej sieci lokalnej. Jest wysoce prawdopodobne, że przy większych odległościach do rozwiązanie z użyciem JMS mogłoby być szybsze, ale nadal byłby to różnice nieznaczne.

Różnice w czasie wykonywania operacji „Dodaj”, „Usuń” czy „Edytuj” biorą się ze specyfiki JMS – serwer dostaje zapytanie od klienta, następnie przesyła zapytanie do serwera JMS, a ten rozsyła informację do wszystkich subskrybentów (w tym do serwera, który wiadomość nadał). Jest to co prawda pewna nadmiarowość jeśli chodzi o wysyłanie wiadomości, ale dzięki takiemu rozwiązaniu zwiększa się ochrona systemu przed niespodziewaną awarią. Przy centralnej bazie danych serwer wysyła zapytanie i od razu dostaje odpowiedź. Nie bierze w tym udziału żaden pośrednik.

Trochę inaczej wygląda sprawa z odczytywaniem istniejących już danych z bazy danych. Tutaj lepsze rezultaty uzyskuje się z użyciem konfiguracji korzystającej z JMS. Gdy klient zapyta na przykład o listę swoich zamówień – aplikacja wyciąga je ze swojej lokalnej bazy danych. Nie ma konieczności wysyłania żadnych wiadomości. W przypadku aplikacji z centralną bazą danych – przy próbie wylistowania zamówień konieczne jest odpytanie zewnętrznej bazy danych, co zwiększa czas przetwarzania żądania.

Otrzymane wyniki pokazują, że zarówno rozwiązanie używające technologii JMS, jak i rozwiązanie z centralną bazą danych mogą znaleźć zastosowanie w obsłudze przedsiębiorstwa handlowego. Wiele tutaj zależy od specyfiki przedsiębiorstwa, od jego wielkości, czy rozmieszczenia geograficznego. Dla przedsiębiorstwa, któremu zależy na odporności na awarie i rozszerzalności – zastosowanie synchronizacji za pomocą JMS jest dobrym rozwiązaniem. Zapewnia one bezproblemowe dołączanie nowych oddziałów, a także sprawia że sklep funkcjonuje nawet wtedy, gdy któryś z oddziałów ma awarię. Z kolei dla przedsiębiorstwa mniejszych rozmiarów, skupionego na kilku oddziałach (znajdujących się raczej w niedalekiej odległości) to rozwiązanie z centralną bazą danych może być najlepsze. Jest prostsze w utrzymaniu i konfiguracji, ale w przypadku awarii bazy danych cały system jest sparaliżowany. Dla przedsiębiorstwa, które posiada kilka oddziałów w różnych miastach, ale obsługują one nie tylko sprzedaż internetową, ale również detaliczną dobrym rozwiązaniem mogłoby być rozwiązanie nie omawiane w tej pracy – każdy oddział posiada bazę danych z przedmiotami, które posiada w magazynie.

Nie udało się niestety przeprowadzić miarodajnych testów obciążeniowych. Konfiguracje sprzętowe, na których były testowane aplikacje bardzo szybko traciły całą moc obliczeniową, przez co uzyskane wyniki nie były w żaden sposób miarodajne. Na sprzęcie przystosowanym do obsługi serwerów prawdopodobnie byłby one o wiele lepsze. Również w takim środowisku można by sprawdzić przepustowość bazy danych i serwera JMS. W tym przypadku braki mocy obliczeniowej i opóźnienia w obsłudze zapytań pojawiały się już przy około 30 jednoczesnych zapytaniach. Jak można zauważyć na wykresach w poprzednim podrozdziale przy 400 zapytaniach czas obsługi zapytania wzrastał z około 1s do maksymalnie 20s. W przypadku systemu internetowego tak długi czas przetwarzania jest niedopuszczalny. Warto zauważyć, że wyszukiwanie zamówień użytkownika zachowywało się podobnie zarówno przy użyciu JMS, jak i centralnej bazy danych. Duże różnice było widać dopiero przy próbie dodania nowego przedmiotu. Tu z powodu dużej liczby wiadomości rozsyłanych między serwerami konfiguracja JMS znacznie spowolniła w stosunku do architektury z centralną bazą danych. Świadczy to o tym, że JMS jest dobry do zastosowania w przedsiębiorstwach, w których nie ma potrzeby obsługi ogromnej liczby użytkowników jednocześnie.

1. Podsumowanie i wnioski

Niniejsza praca przedstawiła podstawy projektowania architektury wielowarstwowej rozproszonej aplikacji internetowej, wspomagającej przedsiębiorstwo handlowe. Na podstawie prostego przykładu aplikacji obsługującej sklep z multimediami przetestowane zostało użycie technologii JMS do komunikacji między serwerami. Pokazane zostały poszczególne etapy tworzenia projektu aplikacji od opisu świata rzeczywistego przez diagramy sekwencji, aż po szczegóły implementacyjne. Przedstawiono również popularne technologie używane we współczesnych systemach internetowych.

Zastosowanie technologii JMS w wielowarstwowych rozproszonych aplikacjach internetowych ma swoje wady i zalety. Jak pokazały testy, rozwiązanie z użyciem JMS jest porównywalne pod względem szybkości w porównaniu z rozwiązaniem z centralną bazą danych. Jest to prawdopodobnie spowodowane tym, że oba rozwiązania potrzebują komunikacji sieciowej do utrzymywania aktualnego statusu bazy danych. Rozwiązanie JMS jest zdecydowanie szybsze jeśli chodzi o bezpośredni dostęp do bazy danych, a dzieje się tak dlatego, że każdy serwer ma swoją własną bazę danych, nie ma potrzeby rozsyłania wiadomości. Jest to też rozwiązanie bezpieczniejsze, bo w przypadku jakiejkolwiek awarii, każdy serwer nadal ma dostępne dane ze swojej lokalnej bazy danych. Przeprowadzone testy pokazały również, że do obsługi systemów z małą i średnią liczbą jednoczesnych użytkowników architektura używająca JMS wydaje się być optymalna. Dużo jednak zależy od specyficznych wymagań, jakie stawiają wobec systemu przedsiębiorcy. Zarówno scentralizowana baza danych, jak i synchronizacja za pomocą JMS mają miejsce we współczesnych przedsiębiorstwach handlowych.

# Literatura

[1] R. Anthony, C. Slominski. Systems Programming: Designing and Developing Distributed Applications, marzec 2015

[2] D. Alur, J. Curpi, D. Malks. Core J2EE Patterns: Best Practices and Design Strategies Second Edition. Prentice Hall 2003

[3] <http://www.oracle.com/technetwork/java/javaee/overview-140548.html>, 10.07.2018r.

[4] B. Scholtz, A. Tijms. The Definitive Guide to JSF in Java EE8: Building Web Applications with JavaServer Faces. Apress, maj 2018

[5] <https://docs.oracle.com/javaee/7/index.html>, 05.07.2018r.

[6] <http://wazniak.mimuw.edu.pl/images/1/1e/ZAWWW-2st1.2-w04.tresc-1.0czb.pdf>, 01.07.2018r.

[7] M. Richards, R. Monson-Haefel, D. A. Chappell. Java Message Service: Creating Distributed Enterprise Applications. O’Reilly Media, czerwiec 2009

[8] R. Rad. Mastering Hibernate. Packt Publishing, maj 2016.