Programowanie aplikacji WWW

Sprawozdanie z projektu I

Temat: „6. Projekt – Wypożyczalnia książek”

Autor: Damian Terlecki

Prowadzący: dr inż. Jacek Grekow

Data: 9-11-2015

1. Założenia projektu

Tematem projektu jest serwis internetowy wypożyczalni książek. Wypożyczalnia umożliwia przeglądanie listy książek, dodawanie książek do koszyka i w efekcie ich wypożyczenie.

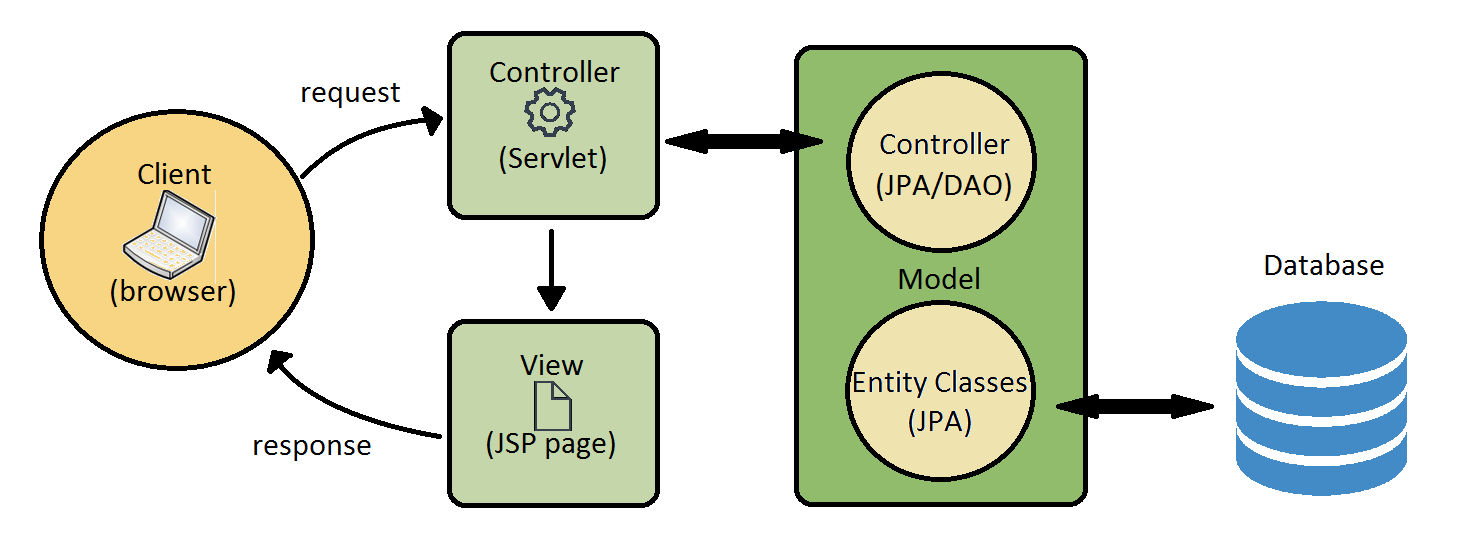
|  |  |
| --- | --- |
| Funkcjonalności | Pkt. |
| 1. Wyświetlanie listy książek. Koszyk z możliwością dodawania i usuwania książek. Możliwość wypożyczenia książek – | 3 |
| 1. Zarządzanie książkami w panelu administracyjnym | 1 |
| 1. Rejestracja czytelników. Potwierdzenie rejestracji w panelu administracyjnym. Możliwość zalogowania się na swoje konto. Tylko użytkownicy zalogowani mogą wypożyczać książki | 2 |
| 1. Konta dla pracowników wypożyczalni. Pracownicy mogą zarządzać książkami, czytelnikami i wypożyczeniami. Pracownikami zarządza administrator | 2 |
| 1. Kategorie książek. Kategorie tworzą wielopoziomowe drzewa. Każda książka należy do dokładnie jednej kategorii, Możliwość przeglądania książek według kategorii | 1 |
| 1. Etykiety (tagi) dla książek. Każda książka może być opatrzona wieloma etykietami. Przeglądanie książek według etykiet | 1 |
| 1. Dodatkowe pliki do pobrania przy opisie ksiąski, np.: kody źródłowe programów, próbki muzyki czy filmów. Każdy plik posiada swój opis. Liczba plików nie jest ograniczona | 1 |
| 1. Wyszukiwanie książek na podstawie tytułu, nazwy autora, numeru ISBN i ewentualnie innych atrybutów | 2 |
| 1. Operatory logiczne typu and, or i not przy wyszukiwaniu | 1 |
| 1. Historia wyszukiwań. Każdy zalogowany użytkownik może zapisać wyniki wyszukiwania w bazie by w innym czasie móc je odtworzyć | 1 |
| 1. Przeglądanie archiwum wypożyczeń czytelnika | 1 |
| 1. Administracja wypożyczeniami. Stany wypożyczeń: książka w magazynie, oczekuje na odbiór (jest na półce czytelnika), wypożyczona. Zmiana stanu wypożyczenia przez pracownika | 1 |
| 1. Stany magazynowe książek. Automatyczna aktualizacja stanów przy wypożyczeniu i zwrocie książki | 1 |
| 1. Możliwość zapisania się do kolejki, gdy danej książki nie ma aktualnie w magazynie | 1 |
| 1. Kary pieniężne za zbyt długie przetrzymywanie książek. Blokada wypożyczeń przy nie zwróceniu książek i niezapłaceniu kary. Maksymalny czas przetrzymywaniu ustawiany przez administratora | 1 |
| 1. Wiadomości redagowane przez administratora i umieszczane na stronie głównej | 1 |
| 1. Wyświetlanie nowości (kilku ostatnio dodanych książek) na stronie głównej | 1 |
| 1. Wybór skórek dla interfejsu użytkownika | 2 |
| 1. Dodatkowa wersja językowa interfejsu użytkownika | 1 |

1. Zrealizowane założenia

W projekcie zrealizowałem wszystkie z wymienionych wyżej założeń.

1. Realizacja

Podczas realizacji projektu kierowałem się następującym schematem aplikacji:

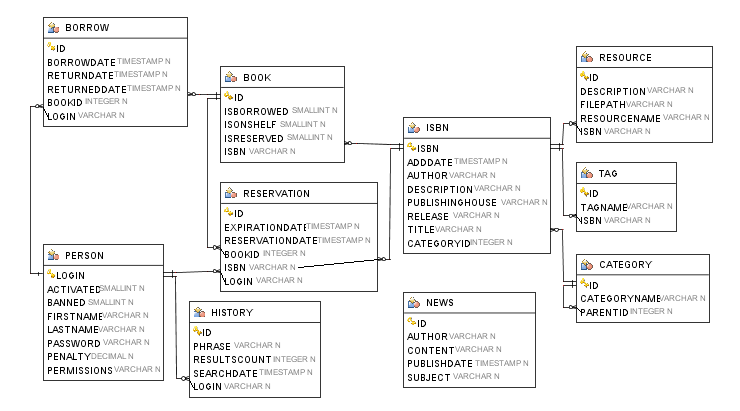


Rysunek 1: Architektura aplikacji

Aplikacja wykorzystuje wzorzec MVC, który dzieli się na trzy główne części:

* Model (JPA) – reprezentuje dane z bazy danych w postaci logicznej,
* Widok (JSP) – wyświetla dane modelu,
* Kontroler (Servlet) – odbiera dane wejściowe od użytkownika przetwarza je, aktualizuje oraz odświeża model.

Zaczynając od warstwy najniższej, wypożyczalnia książek korzysta z następującej bazy danych:

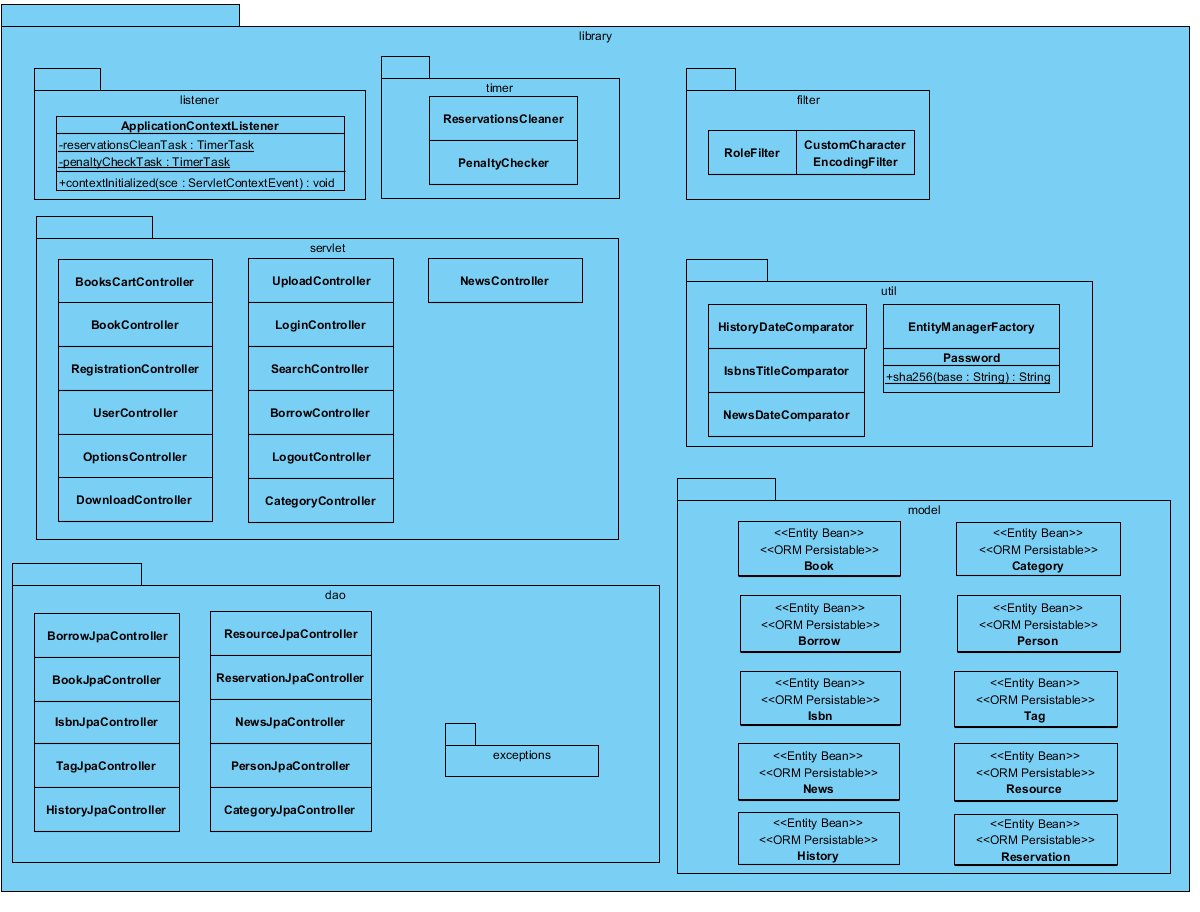


Rysunek 2: Schemat bazy danych

Pewne założenia wymagające wyjaśnień:

1. Jeśli książka nie ma numeru ISBN (co się zdarza, choć bardzo rzadko) to należy nadać jej unikalną wartość gdyż książki rozróżniane są po polu ISBN.
2. Egzemplarze tych samych książek nie różnią się numerem ISBN (różnią się numerem id).
3. Do numeru ISBN przypisane są kategoria, tagi i zasoby (np. pliki) oraz rezerwacje w przypadku braku książek w magazynie.
4. Do książki przypisane są wypożyczenia oraz rezerwacje rozumiane jako czekające na odbiór przez klienta.

Kolejne warstwy – modelu i kontrolerów oraz klasy pomocnicze przedstawia diagram klas i pakietów aplikacji:

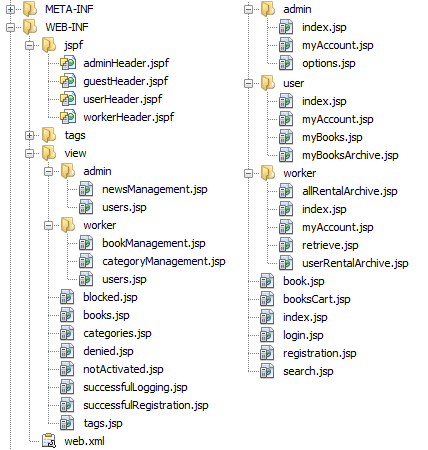


Rysunek 3: Diagram klas i pakietów

Fragmenty tych dwóch warstw warte zwrócenia uwagi:

* Hasła haszowane funkcją skrótu SHA-256, w kolejnych wersjach aplikacji mogłyby być uzupełnione o sól/sole.
* EntityManagerFactory – synchronizowany singleton tworzony na podstawie konfiguracji połączenia z bazą danych (plik presistence.xml). Tworzy instancje EntityManager, z których korzystają kontrolery JPA do komunikacji z bazą danych. Typ transakcji to: resource\_local.
* PenaltyChecker i ReservationCleaner to klasy dziedziczące z TimerTask. Odpowiadają za usuwanie rezerwacji po terminie z bazy danych, przydzielanie rezerwacji użytkownikom w kolejce oraz blokowanie użytkowników, którzy nie zwrócili książek i minął już termin. Zadania te są inicjowane, uruchamiane cyklicznie i usuwane w klasie ApplicationContextListener dziedziczącej z ServletContextListener.
* ApplicationContextListener inicjuje kontekst aplikacji z pliku (np. dane związane z maksymalną liczbą dni dla wypożyczenia czy liczba wiadomości wyświetlanych na stronie głównej).
* RoleFilter – filtr przekierowujący gości i użytkowników w przypadku braku uprawnień oraz zablokowanego lub nieaktywowanego konta.
* CustomCharacterEncodingFilter – filtr ustawiający kodowanie zapytania i odpowiedzi na „UTF-8”, umożliwia poprawne przesyłanie znaków metodą post. W przypadku metody Get wymagana jest dodatkowa konfiguracja połączenia servera (w przypadku Tomcata).

Strukturę ostatniej warstwy – widokowej przedstawia drzewo strony:



Rysunek 4: Struktura warstwy widokowej

Wydzielony został podfolder „view” w folderze WEB-INF. W ten sposób użytkownik nie ma dostępu do dynamicznych widoków w inny sposób niż przez kontroler. Dla pozostałych stron w przypadku uzyskania dostępu dane będą już załadowane. Podział struktury na foldery takie jak „admin”, „worker” czy „user” pozwala na nieco inne zarządzanie dostępem do widoków niż przez servlety. Dostęp regulowany jest w filtrze RoleFilter.

Cztery różne nagłówki (.jspf) w zależności od roli dodawane są na podstawie uprawnień zalogowanego użytkownika. Pliki z rozszerzeniem .tag używane są między innymi do rekurencyjnego wypisywania kategorii.

1. Fragmenty kodu
2. Dodatkowa wersja językowa interfejsu (l18n):

Przy implementacji tego założenia skorzystałem z biblioteki JSTL a w szczególności z tagów fmt: setLocale i setBundle, dzięki którym włączam pliki z tłumaczeniem do strony. Umożliwia to korzystanie z tagu message z parametrem key – odwzorowaniem z pliku na przetłumaczoną frazę.

<c:set var="language" value="${not empty param.language ? param.language : not empty language ? language : pageContext.request.locale}" scope="session" />

<fmt:setLocale value="${language}" />

<fmt:setBundle basename="library.i18n.text" />

<form>

<select id="language" name="language" onchange="submit()" width="90" style="width: 90px">

<option value="pl" ${language == 'pl' ? 'selected' : ''}><fmt:message key="polish"/></option>

<option value="en" ${language == 'en' ? 'selected' : ''}><fmt:message key="english"/></option>

</select>

</form>

Fragmenty plików z tłumaczeniami:

text\_pl.properties:

polish = Polski

english = Angielski

validation.blank = Te pole nie może być puste

validation.confirmPassword = Potwierdzenie hasła niepoprawne

text\_en.properties:

polish = Polish

english = English

validation.blank = This field cannot be empty

validation.confirmPassword = Incorrect password confirmation

W ten sposób możliwe jest szybkie wprowadzenie dodatkowych języków do strony – wystarczy przetłumaczyć plik i dodać kolejną opcję wyboru języka. W następnych wersjach aplikacji można wprowadzić zapisywanie i odczytywanie wyboru języka oraz skórki z cookie.

1. Importowanie wydzielonego nagłówka:

<%@ include file="/WEB-INF/jspf/userHeader.jspf" %>

1. Rekurencyjne wypisywanie kategorii (plik categories.tag):

<%@ attribute name="list" type="java.util.Vector" required="true" %>

<c:forEach var="category" items="${list}">

<li><input type="radio" name="selectedCategory" value="${category.id}"

<c:if test="${not empty isbnInDb && not empty isbnInDb.isbn.category && isbnInDb.isbn.category.id eq category.id}">checked='checked'</c:if>>${category.categoryName}</li>

**<myTags:categories list="${category.categoryList}"/>**

</c:forEach>

1. Filtr uprawnień:

//…

if (user != null && user.isBanned() && !url.contains("LogoutController") && !url.contains("myAccount.jsp") && !url.contains("myBooks.jsp")) {

((HttpServletResponse) response).sendRedirect("/Library/blocked.jsp");

}

if (user != null && !user.isActivated() && !url.contains("/myAccount.jsp") && !url.contains("/index.jsp")) {

request.getRequestDispatcher("/WEB-INF/view/notActivated.jsp").forward(request, response);

return;

}

//…

if (url.contains("/admin/")) {

if (user == null || !user.getPermissions().equals("ADMIN")) {

if (user == null) {

session.setAttribute("redirectPath", "/Library" + url);

((HttpServletResponse) response).sendRedirect("/Library/login.jsp");

} else if (!url.contains("LogoutController") && !url.contains("myAccount.jsp") && !url.contains("myBooks.jsp")) {

request.getRequestDispatcher("/WEB-INF/view/denied.jsp").forward(request, response);

} else {

chain.doFilter(request, response);

}

} else {

chain.doFilter(request, response);

}

}

//…

1. Implementacja wyszukiwania:

Fraza zostaje sparsowana do zapytania SQL, które jest następnie wywołane:

public List<Isbn> findIsbnByQuery(String query) {

EntityManager em = getEntityManager();

try {

String wholeQuery = "SELECT \* FROM APP.ISBN WHERE " + query;

Query runQuery = em.createNativeQuery(wholeQuery, Isbn.class);

return runQuery.getResultList();

} finally {

em.close();

}

}

1. Obsługa dodawania wiadomości w kontrolerze:

//…

News news = new News();

news.setSubject(request.getParameter("subject"));

news.setAuthor(request.getParameter("author"));

news.setContent(request.getParameter("body"));

try {

news.setPublishDate(formatter.parse(request.getParameter("date")));

} catch (ParseException ex) {

Logger.getLogger(NewsController.class.getName()).log(Level.SEVERE, null, ex);

request.getSession().setAttribute("error", "dateFormat");

}

newsDao.create(news);

//…

1. Filtr zapewniający poprawne przesyłanie znaków metodą post:

request.setCharacterEncoding("UTF-8");

response.setCharacterEncoding("UTF-8");

chain.doFilter(request, response);

1. Cykliczne zadania uruchomione przy starcie aplikacji (contextInitialized(…)):

@Override

public void contextInitialized(ServletContextEvent sce) {

//…

ReservationsCleaner.max\_receive\_wait\_time = Integer.parseInt((String) sce.getServletContext().getAttribute("max\_receive\_wait\_time"));

reservationsCleanTask = new ReservationsCleaner();

Timer reservationCleanTimer = new Timer();

reservationCleanTimer.schedule(reservationsCleanTask, 0, (1 \* 60 \* 60 \* 1000));

penaltyCheckTask = new PenaltyChecker();

Timer penaltyCheckTimer = new Timer();

reservationCleanTimer.schedule(penaltyCheckTask, 0, (1 \* 60 \* 60 \* 1000));

@Override

public void contextDestroyed(ServletContextEvent sce) {

try {

reservationsCleanTask.cancel();

} catch (Exception ex) {

Logger.getLogger(ApplicationContextListener.class.getName()).log(Level.SEVERE, null, ex);

}

try {

penaltyCheckTask.cancel();

} catch (Exception ex) {

Logger.getLogger(ApplicationContextListener.class.getName()).log(Level.SEVERE, null, ex);

}

}

1. Zainicjowanie konta administratora w przypadku pustej bazy:

if (userDao.findPersonEntities().isEmpty()) {

Person admin = new Person();

admin.setActivated(true);

admin.setBanned(false);

admin.setPassword(Password.sha256("admin"));

admin.setFirstName("admin");

admin.setLastName("admin");

admin.setPermissions("ADMIN");

try {

userDao.create(admin);

} catch (Exception ex) {

Logger.getLogger(ApplicationContextListener.class.getName()).log(Level.SEVERE, null, ex);

}

}

1. Inne:

* Jako edytor wiadomości wykorzystałem edytor HTML TinyMCE na licencji open source LGPL.
* Skórki CSS zostały stworzone przy użyciu bootstrapa na licencji MIT.