Programowanie aplikacji WWW

Sprawozdanie z projektu II

Temat: „6. Projekt – Wypożyczalnia książek”

Autor: Damian Terlecki

Prowadzący: dr inż. Jacek Grekow

Data: 13-12-2015

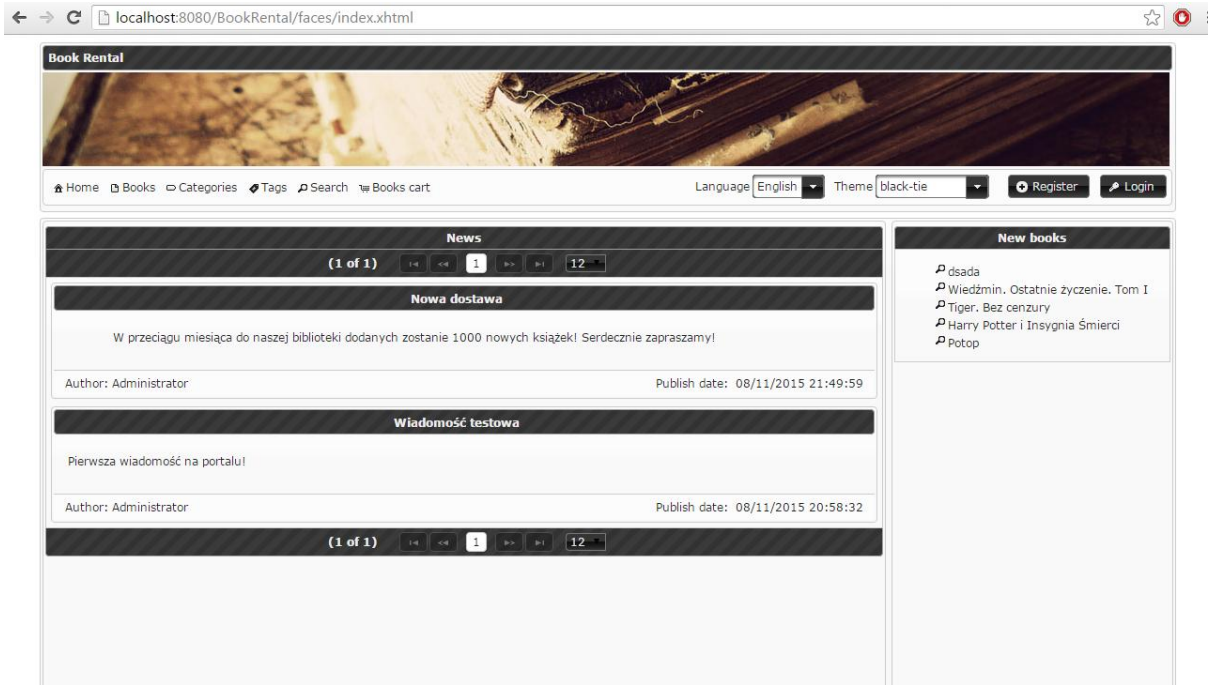
1. Założenia projektu

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

Funkcjonalności	Pkt.
1. Wyświetlanie listy książek. Koszyk z możliwością dodawania i usuwania książek. Możliwość wypożyczenia książek –	3
2. Zarządzanie książkami w panelu administracyjnym	1
3. Rejestracja czytelników. Potwierdzenie rejestracji w panelu administracyjnym. Możliwość zalogowania się na swoje konto. Tylko użytkownicy zalogowani mogą wypożyczać książki	2
4. Konta dla pracowników wypożyczalni. Pracownicy mogą zarządzać książkami, czytelnikami i wypożyczeniami. Pracownikami zarządza administrator	2
5. Kategorie książek. Kategorie tworzą wielopoziomowe drzewa. Każda książka należy do dokładnie jednej kategorii, Możliwość przeglądania książek według kategorii	1
6. Etykiety (tagi) dla książek. Każda książka może być opatrzona wieloma etykietami. Przeglądanie książek według etykiet	1
7. 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
8. Wyszukiwanie książek na podstawie tytułu, nazwy autora, numeru ISBN i ewentualnie innych atrybutów	2
9. Operatory logiczne typu and, or i not przy wyszukiwaniu	1
10. Historia wyszukiwań. Każdy zalogowany użytkownik może zapisać wyniki wyszukiwania w bazie by w innym czasie móc je odtworzyć	1
11. Przeglądanie archiwum wypożyczeń czytelnika	1
12. Administracja wypożyczeniami. Stany wypożyczeń: książka w magazynie, oczekuje na odbiór (jest na półce czytelnika), wypożyczona. Zmiana stanu wypożyczenia przez pracownika	1
13. Stany magazynowe książek. Automatyczna aktualizacja stanów przy wypożyczeniu i zwrocie książki	1
14. Możliwość zapisania się do kolejki, gdy danej książki nie ma aktualnie w magazynie	1
15. Kary pieniężne za zbyt długie przetrzymywanie książek. Blokada wypożyczeń przy nie zwróceniu książek i niezapłaceniu kary. Maksymalny czas przetrzymywaniu ustawiany przez administratora	1
16. Wiadomości redagowane przez administratora i umieszczane na stronie głównej	1
17. Wyświetlanie nowości (kilku ostatnio dodanych książek) na stronie głównej	1
18. Wybór skórek dla interfejsu użytkownika	2
19. Dodatkowa wersja językowa interfejsu użytkownika	1

2. Zrealizowane założenia

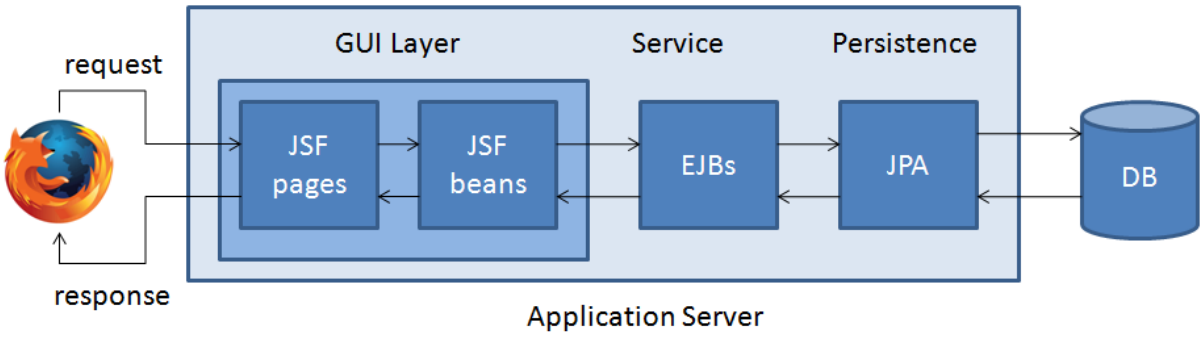
W projekcie zrealizowałem wszystkie z wymienionych wyżej założeń.



Rysunek 1: Wygląd serwisu

3. Realizacja

Podczas realizacji projektu kierowałem się następującym schematem aplikacji:



Rysunek 2: Architektura aplikacji

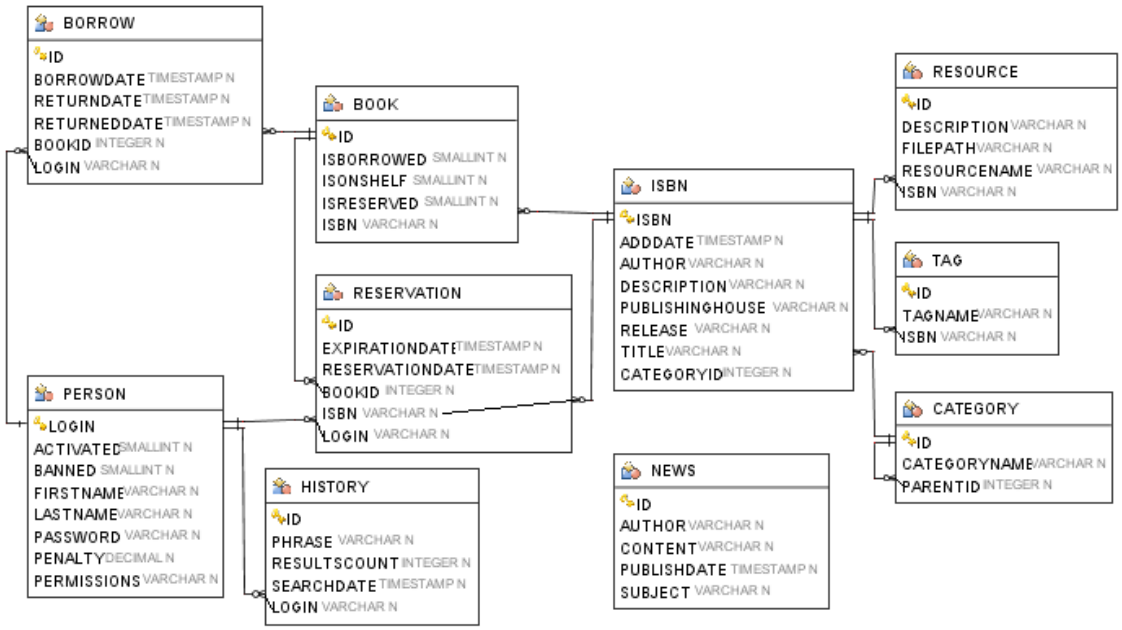
Źródło: http://www.thejavageek.com/wp-content/uploads/2015/01/application-architecture.png

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

- Model (JPA + EJB) – reprezentuje dane z bazy danych w postaci logicznej,
- Widok (JSF page) – wyświetla dane modelu,
- Kontroler (JSF bean) – odbiera dane wejściowe od użytkownika przetwarza je, aktualizuje oraz odświeża model.

Warstwa serwisowa w tym przypadku realizuje jedynie podstawowe operacje CRUD.

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

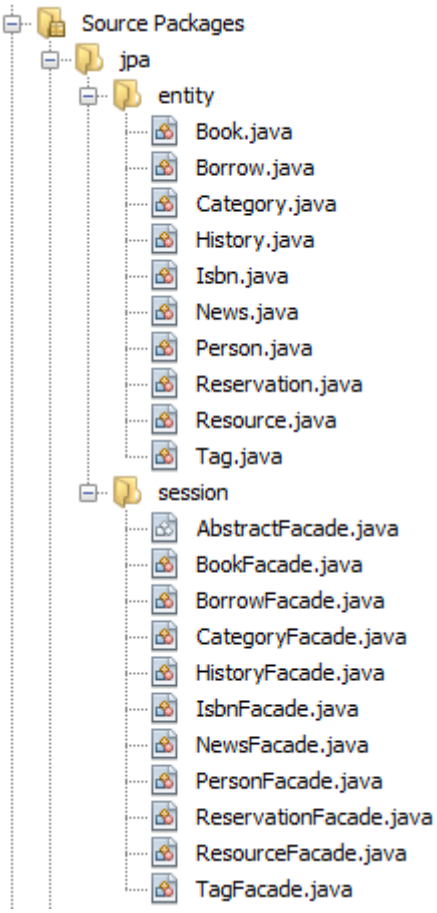


Rysunek 3: Schemat bazy danych

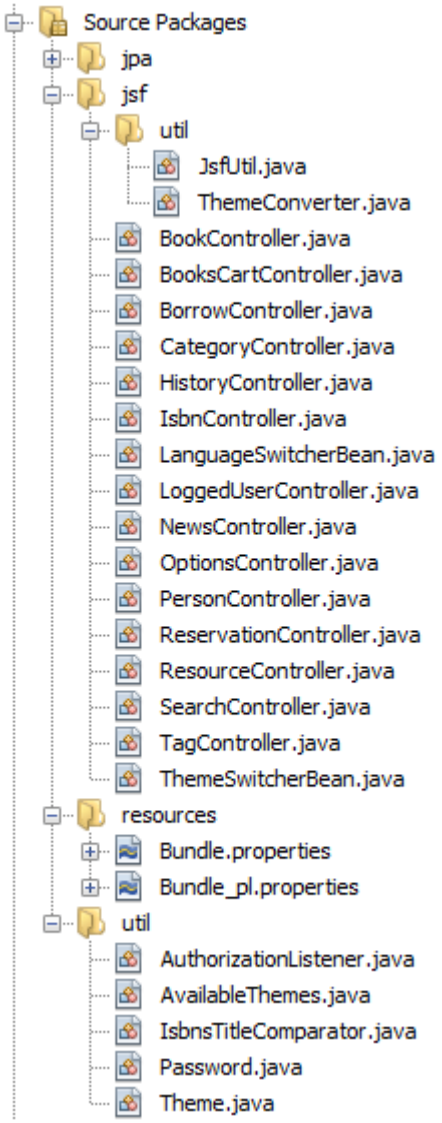
Pewne założenia wymagające wyjaśnień:

a) 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.
b) Egzemplarze tych samych książek nie różnią się numerem ISBN (różnią się numerem id).
c) Do numeru ISBN przypisane są kategoria, tagi i zasoby (np. pliki) oraz rezerwacje w przypadku braku książek w magazynie.
d) Do książki przypisywane są wypożyczenia oraz rezerwacje.

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



Rysunek 4: Klasy i pakiety modelu (entity – JPA, session - EJB)



Rysunek 5: Pakiet kontrolerów (jsf - Managed Bean), resources - pliki językowe

Klasy entity - POJO, opakowują dane pobierane z bazy danych w typy abstrakcyjne np. Book.

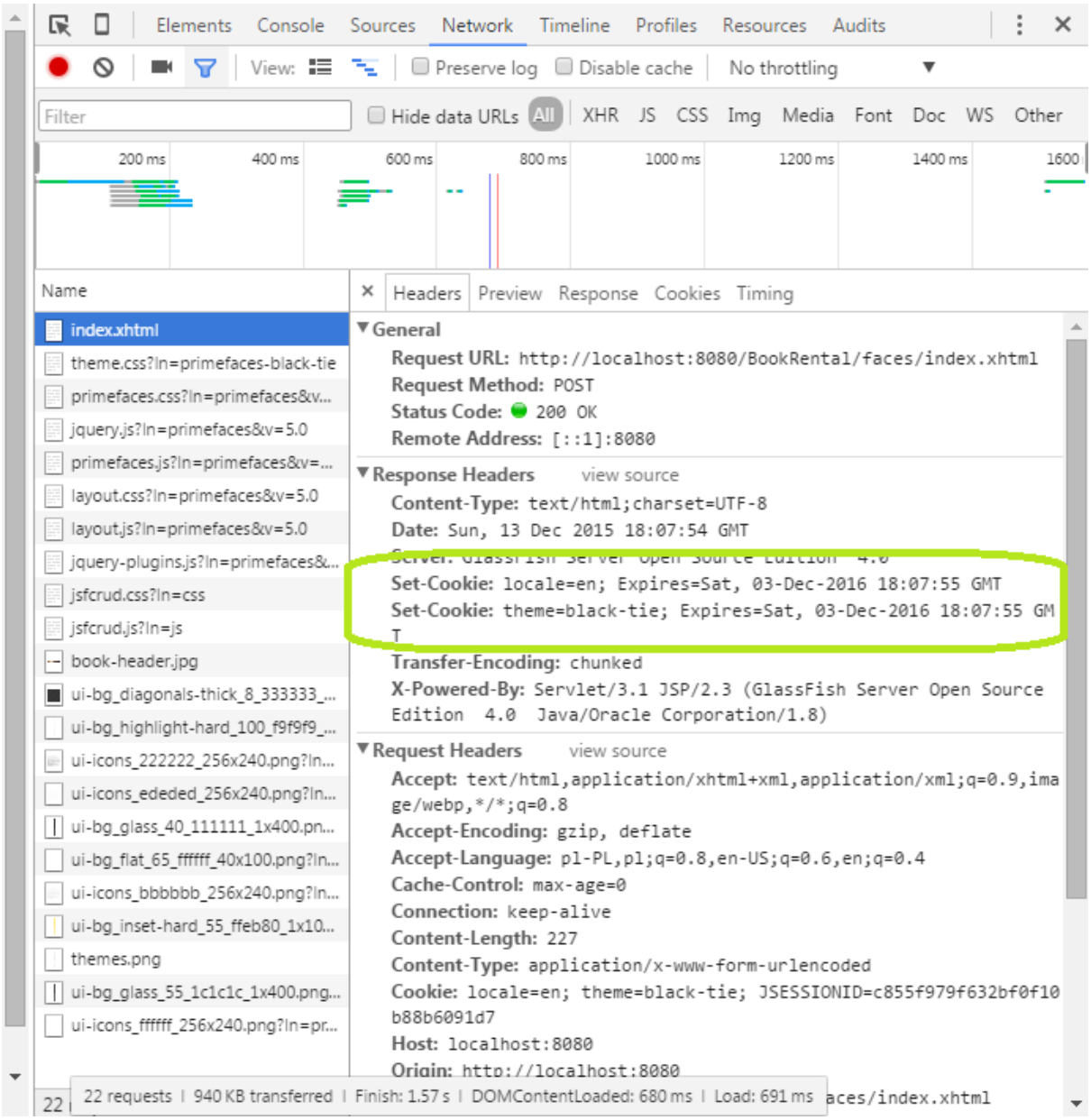
Klasy session / facade – fasady EJB, pozwalają na komunikację z bazą danych na wyższym poziomie niż JDBC. Beany te działają po stronie serwera i mogą być wstrzykiwane do innych elementów. W tym przypadku adnotowane @Stateless oznaczane jako bezstanowe, pozwalają na implementację serwisów.

Klasy controller / jsf – kontrolery widoków JSF, beany zarządzane (ManagedBean), z deklarowanym zakresem np. SessionScoped, ViewScoped, ApplicationScoped, itp. Przechowują stan zgodnie z zakresem i zapewniają komunikację ze stronami JSF. Strony JSF mogą pobierać stan beana za pomocą wyrażenia #{bean.field} (przy założeniu, że zaimplementowano gettery i settery).

Fragmenty tych dwóch warstw warte zwrócenia uwagi:

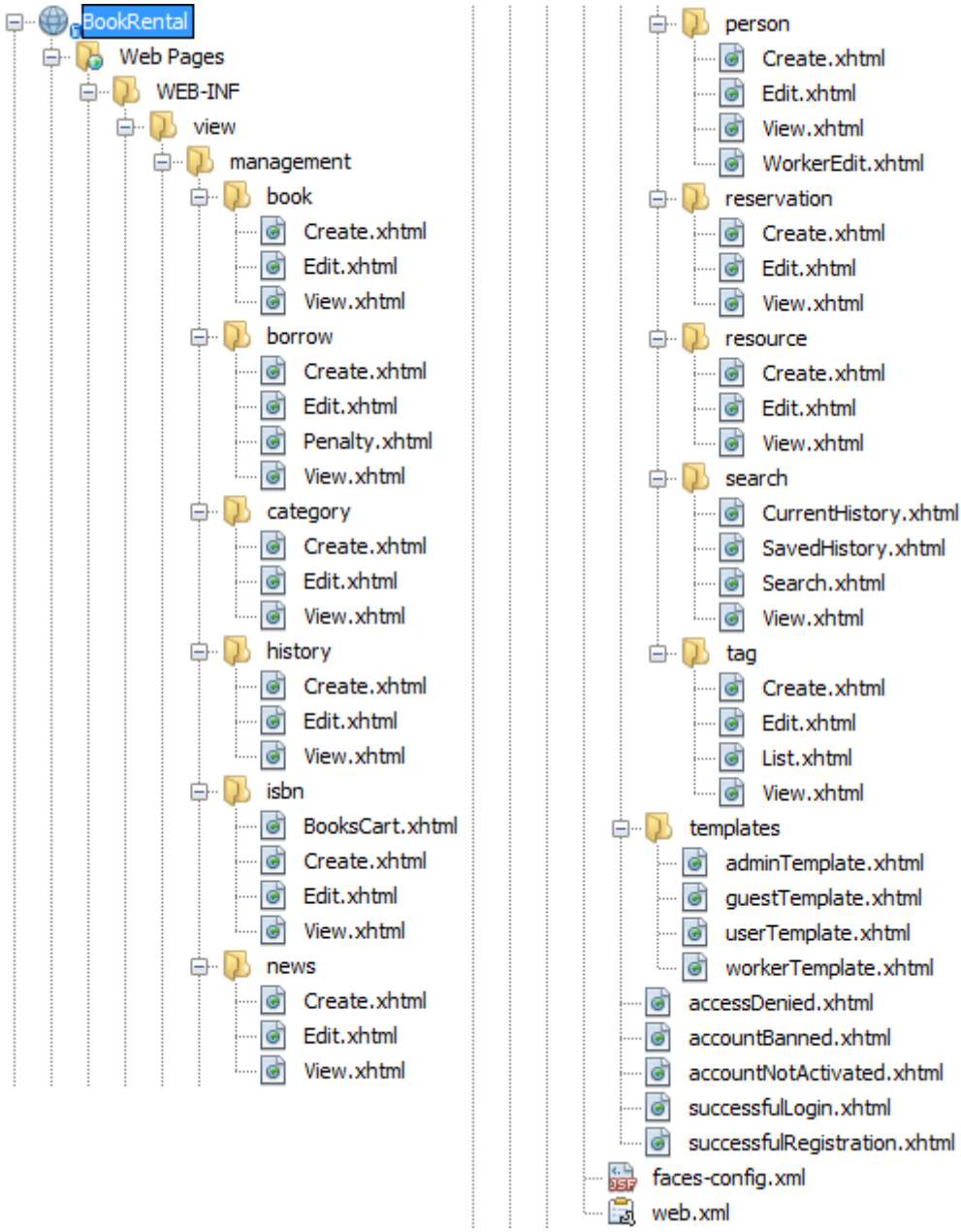
- Hasła haszowane funkcją skrótu SHA-256, w kolejnych wersjach aplikacji mogłyby być uzupełnione o sól/sole.

- OptionsController ładuje opcje serwisu z pliku na serwerze (np. dane związane z maksymalnym okresem wypożyczenia czy wartością kary za dzień spóźnienia).
- AuthorizationFilter – phaseFilter nadzorujący dostępen do stron wymagających określonych uprawnień. Odpowiada za przekierowania w przypadku, gdy nie jest aktywowane lub zostało zbanowane.
- ThemeSwitcherBean oraz LanguageSwitcherBean zapisują do ciasteczek wybór preferencji dotyczących języka i skórki wybranych przez użytkownika



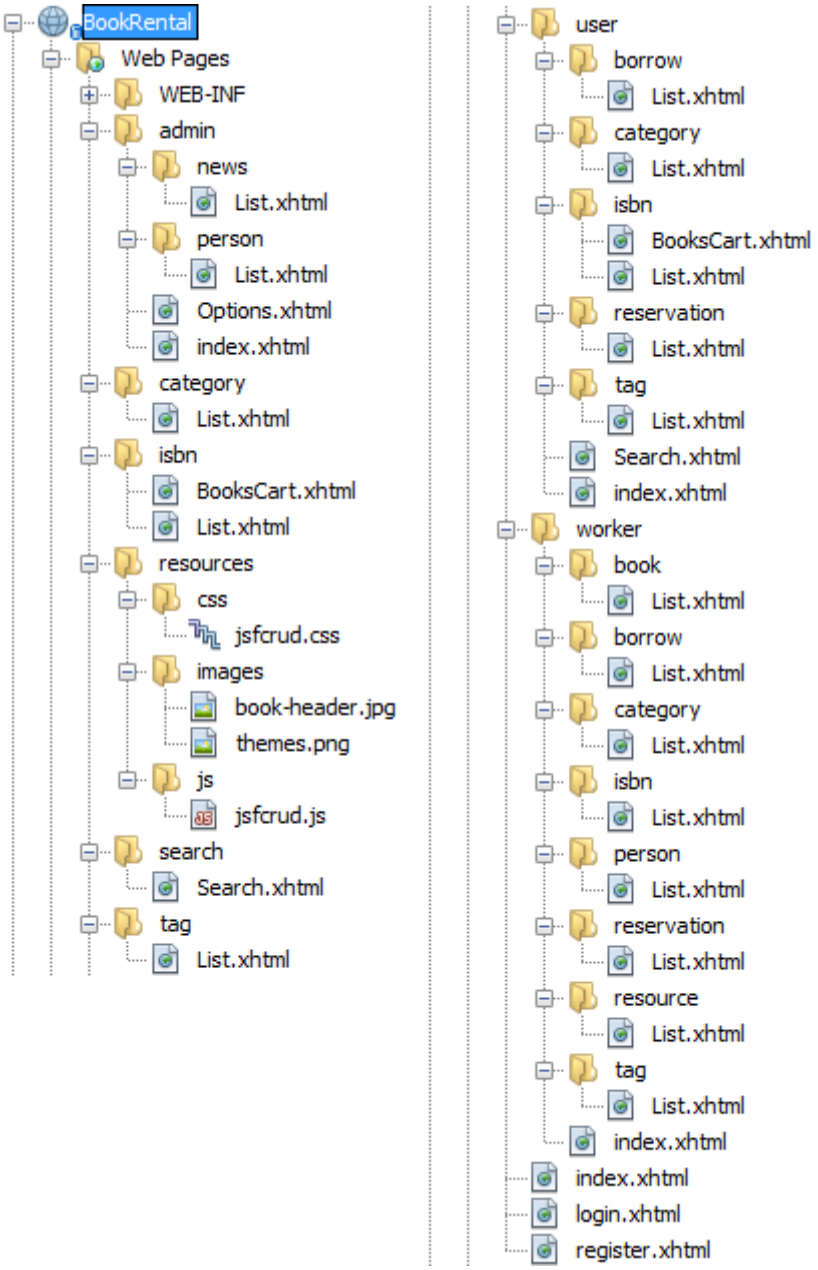
Rysunek 6: Ciasteczka w odpowiedzi HTTP

Na strukturę ostatniej warstwy – widokowej składają się strony JSF. Do budowania stron JSF wykorzystywany jest język Facelets. W przypadku tego projektu wykorzystana została również open source'owa biblioteka komponentów UI – PrimeFaces. Dzięki temu framewrokowi możliwe jest tworzenie bogatszego interfejsu użytkownika.



Rysunek 7: Struktura warstwy widokowej, folder WEB-INF

WEB-INF to specjalny folder w strukturze aplikacji, który nie jest dostępny publicznie, nie jest częścią publicznego drzewa dokumentu. Wydzielony został podfolder „view” w folderze WEB-INF a w nim konkretne widoki. Elementy o charakterystycznych nazwach takich jak „Create”, „Edit” czy „View” reprezentują okna (dialog), które poprzez kompozycje włączane są do innych widoków (głównie list) i wyświetlane podczas kliknięcia przycisku. Dla użytkownika dostęp do okien jest sensowny jedynie poprzez widoki, dlatego też zostały one umieszczone w folderze WEB-INF. Kolejnymi elementami są szablony, które również korzystają z mechanizmu kompozycji i użytkownik nie powinien mieć do nich dostępu. Ostatnią częścią składową folderu WEB-INF są strony, do których użytkownik zostaje przekierowany po zalogowaniu się. Informują one między innymi o braku praw do przeglądania strony, zablokowanym czy nieaktywnym koncie, jak też o powodzeniu procesu logowania lub rejestracji.



Rysunek 8: Struktura warstwy widokowej, strony dostępne użytkownikom

Cztery różne szablony podpinane są do stron w zależności od praw dostępu zalogowanego (bądź nie) użytkownika. Wyświetlane dane również się zmieniają, dlatego utworzone zostały różne widoki wyświetlające dane w postaci list. Widoki te korzystają ze wspólnych okien zadeklarowanych w folderze WEB-INF.

4. Fragmenty kodu

a) Dostęp do bazy danych:

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.

```
//glassfish-resources.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource
Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="false"
connection-creation-retry-attempts="0" connection-creation-retry-interval-in-seconds="10" connection-
leak-reclaim="false" connection-leak-timeout-in-seconds="0" connection-validation-method="auto-commit"
datasource-classname="org.apache.derby.jdbc.ClientDataSource" fail-all-connections="false" idle-
timeout-in-seconds="300" is-connection-validation-required="false" is-isolation-level-
guaranteed="true" lazy-connection-association="false" lazy-connection-enlistment="false" match-
connections="false" max-connection-usage-count="0" max-pool-size="32" max-wait-time-in-millis="60000"
name="derby_net_library_appPool" non-transactional-connections="false" pool-resize-quantity="2" res-
type="javax.sql.DataSource" statement-timeout-in-seconds="-1" steady-pool-size="8" validate-atmost-
once-period-in-seconds="0" wrap-jdbc-objects="false">
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="1527"/>
        <property name="databaseName" value="library"/>
        <property name="User" value="app"/>
        <property name="Password" value="app"/>
        <property name="URL" value="jdbc:derby://localhost:1527/library"/>
        <property name="driverClass" value="org.apache.derby.jdbc.ClientDriver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/library" object-type="user" pool-
name="derby_net_library_appPool"/>
</resources>


//persistence.xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="BookRentalPU" transaction-type="JTA">
    <jta-data-source>jdbc/library</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Zarządzaniem połączeniem z bazą danych zajmuje się server Glassfish. Typ bazy danych to Java DB. Typ transakcji to JTA, EntityManager jest wstrzykiwany poprzez @PersistenceContext.

b) Przykładowy kontroller (JSF Managed Bean)

Bean realizujący zmianę język, korzystający z mechanizmu ciasteczek:

```
@ManagedBean(eager = true)
@SessionScoped
public class LanguageSwitcherBean implements Serializable {
/*…*/
    @PostConstruct
    public void init() {
        ExternalContext externalContext = FacesContext.getCurrentInstance().getExternalContext();
        Map<String, Object> cookies = externalContext.getRequestCookieMap();
        Cookie cookie = (Cookie) cookies.get("locale");
        if (cookie != null) {
            locale = cookie.getValue();
        }
        for (Map.Entry<String, Object> entry : countries.entrySet()) {
            if (entry.getValue().toString().equals(locale)) {
                FacesContext.getCurrentInstance()
                        .getViewRoot().setLocale((Locale) entry.getValue());
            }
        }
    }

    public void localeChanged(ValueChangeEvent e) {
        String newLocaleValue = e.getNewValue().toString();
        for (Map.Entry<String, Object> entry : countries.entrySet()) {
            if (entry.getValue().toString().equals(newLocaleValue)) {
                FacesContext.getCurrentInstance()
                        .getViewRoot().setLocale((Locale) entry.getValue());
                Map<String, Object> properties = new HashMap<>();
                properties.put("maxAge", new Integer(30758400));    //365 days
                FacesContext.getCurrentInstance()
                        .getExternalContext()
                        .addResponseCookie("locale", locale, properties);
            }
        }
    }
}

//faces-config.xml
    <application>
        <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>pl</supported-locale>
        </locale-config>
        <resource-bundle>
            <base-name>/resources/Bundle</base-name>
            <var>bundle</var>
        </resource-bundle>
    </application>
```

Po skonstruowaniu beana, a przed jego użyciem ustawiony zostanie język (na podstawie ciasteczka lub wcześniej zainicjowanej wartości w przypadku braku ciasteczka). Po zmianie języka tekst wczytywany będzie z odpowiedniego pliku Bundle* z katalogu resources.

c) Autoryzacja użytkownika:

Aby uniknąć ustawiania domen użytkowników po stronie serwera, zastosowany został PhaseFilter. Dzięki niemu użytkownik będzie przekierowywany w przypadku nieodpowiednich uprawnień.

```
//faces-config.xml
    <lifecycle>
        <phase-listener>util.AuthorizationListener</phase-listener>
    </lifecycle></c:forEach>
```

```
//AuthorizationListener
public class AuthorizationListener implements PhaseListener {

    @Override
    public void afterPhase(PhaseEvent event) {

        FacesContext facesContext = event.getFacesContext();
        String currentPage = facesContext.getViewRoot().getViewId();

        HttpSession session = (HttpSession) facesContext.getExternalContext().getSession(true);
        Person loggedUser = (Person) session.getAttribute("user");
        if (loggedUser == null) {
            if (currentPage.contains("/user/") || currentPage.contains("/worker/") ||
currentPage.contains("/admin/")) {
                redirectToLogin(facesContext);
            }
        } else {
            if (currentPage.contains("/worker/")) {
                if ("USER".equals(loggedUser.getPermissions())) {
                    redirectAccessDenied(facesContext);
                    return;
                }
            } else if (currentPage.contains("/admin/")) {
                if (!"ADMIN".equals(loggedUser.getPermissions())) {
                    redirectAccessDenied(facesContext);
                    return;
                }
            }

            if (!loggedUser.getActivated()) {
                redirectNotActivated(facesContext);
                return;
            }

            if (loggedUser.getBanned() && !currentPage.contains("/user/borrow/") &&
!currentPage.contains("/user/reservation/")) {
                redirectBanned(facesContext);
                return;
            }
        }
    }
/*…*/
}
```

d) Kolejny przykład kontrolera (Managed Bean):

```
@ManagedBean(name = "categoryController")

@ViewScoped

public class CategoryController implements Serializable {


            @EJB
            private jpa.session.CategoryFacade ejbFacade;
            private List<Category> items = null;
            private Category selected;
            private TreeNode root;

/*…*/
```

```
private void persist(PersistAction persistAction, String successMessage) {
        if (selected != null) {
            setEmbeddableKeys();
            try {
                if (persistAction != PersistAction.DELETE) {
                    getFacade().edit(selected);
                } else {
                    getFacade().remove(selected);
                }
                if (successMessage != null) {
                    JsfUtil.addSuccessMessage(successMessage);
                }
            } catch (EJBException ex) {
                String msg = "";
                Throwable cause = ex.getCause();
                if (cause != null) {
                    msg = cause.getLocalizedMessage();
                }
                if (msg.length() > 0) {
                    JsfUtil.addErrorMessage(msg);
                } else {
                    JsfUtil.addErrorMessage(ex,
ResourceBundle.getBundle("/resources/Bundle").getString("PersistenceErrorOccured"));
                }
            } catch (Exception ex) {
                Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);
                JsfUtil.addErrorMessage(ex,
ResourceBundle.getBundle("/resources/Bundle").getString("PersistenceErrorOccured"));
            }
        }
    }

/*gettery I settery
...
*/
```

Na początku klasy widoczna jest deklaracja fasady wraz z adnotacją @EJB. Dzięki adnotacji fasada zostanie wstrzygnięta (dependency injection) do beana. Wszystkie fasady mają adnotację @Stateless, to znaczy są bezstanowe. Relizują jedynie dostęp do bazy danych. Główną klasą, z której dziedziczą fasady jest AbstractFacade:

```
public abstract class AbstractFacade<T> {
    protected Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }
```

```
    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }

}
```

e) Przykładowa strona JSF (lista rezerwacji):

Na początku strony widoczne jest wykorzystanie kompozycji (szablon), następnie zdefiniowany jest nagłówek i ciało, w którym znajduje się lista rezerwacji. Na końcu do strony włączony widok – okno (dialog). Po wybraniu rezerwacji i kliknięciu w przycisk View ukazuje się okno z informacjami o wybranej rezerwacji.

```
    <ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
                xmlns:h="http://xmlns.jcp.org/jsf/html"
                xmlns:f="http://xmlns.jcp.org/jsf/core"
                xmlns:p="http://primefaces.org/ui"
                template="/WEB-INF/view/templates/userTemplate.xhtml">


    <ui:define name="title">
        <h:outputText value="#{bundle.ListReservationTitle}"></h:outputText>
    </ui:define>

    <ui:define name="body">
        <h:form id="ReservationListForm">
            <p:panel header="#{bundle.ListReservationTitle}">
                <p:dataTable id="datalist" value="#{reservationController.userItems}" var="item"
                             selectionMode="single" selection="#{reservationController.selected}"
                             paginator="true"
                             rowKey="#{item.id}"
                             rows="10"
                             rowsPerPageTemplate="10,20,30,40,50"
                             >
```

```
                    <p:ajax event="rowSelect"   update="viewButton deleteButton"/>
                    <p:ajax event="rowUnselect" update="viewButton deleteButton"/>

                    <p:column>
                        <f:facet name="header">
                            <h:outputText value="#{bundle.ListReservationTitle_id}"/>
                        </f:facet>
                        <h:outputText value="#{item.id}"/>
                    </p:column>
                    <p:column>
                        <f:facet name="header">
                            <h:outputText value="#{bundle.ListReservationTitle_placeInQueue}"/>
                        </f:facet>
                        <h:outputText value="#{reservationController.getQueuePlace(item)}"/>
                    </p:column>
                    <p:column>
                        <f:facet name="header">
                            <h:outputText value="#{bundle.ListReservationTitle_reservationDate}"/>
                        </f:facet>
                        <h:outputText value="#{item.reservationDate}">
                            <f:convertDateTime pattern="dd/MM/yyyy HH:mm:ss" />
                        </h:outputText>
                    </p:column>
                    <p:column>
                        <f:facet name="header">
                            <h:outputText value="#{bundle.ListReservationTitle_isbn}"/>
                        </f:facet>
                        <h:outputText value="#{item.isbn}"/>
                    </p:column>
                    <p:column>
                        <f:facet name="header">
                            <h:outputText value="#{bundle.ListReservationTitle_person}"/>
                        </f:facet>
                        <h:outputText value="#{item.person}"/>
                    </p:column>
                    <f:facet name="footer">
                        <p:commandButton id="viewButton"   icon="ui-icon-search"
value="#{bundle.View}" update=":ReservationViewForm" oncomplete="PF('ReservationViewDialog').show()"
disabled="#{empty reservationController.selected}"/>
                        <p:commandButton id="deleteButton" icon="ui-icon-trash"
value="#{bundle.Delete}" actionListener="#{reservationController.destroy}" update=":growl,datalist"
disabled="#{empty reservationController.selected}"/>
                    </f:facet>
                </p:dataTable>
            </p:panel>
        </h:form>

        <ui:include src="/WEB-INF/view/management/reservation/View.xhtml"/>
    </ui:define>
</ui:composition>
```

f) Przykładowy szablon (szablon dla administratora):

Dzięki znacznikowi <ui:insert> realizowany jest mechanizm szablonu. W te mejsca możliwe jest opcjonalne włączenie części strony.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
```

```
    <h:head>
        <title><ui:insert name="title">#{bundle.SiteTitle} - (#{bundle.Admin})</ui:insert></title>
        <h:outputStylesheet library="css" name="jsfcrud.css"/>
        <h:outputScript library="js" name="jsfcrud.js"/>
        <h:outputStylesheet library="primefaces-#{themeSwitcherBean.pickedTheme}" name="theme.css" />
    </h:head>

    <h:body>

        <p:growl id="growl" life="5000" />

        <p:layout fullPage="false" style="width: 1210px; display: block; margin: 0 auto;" >

            <p:layoutUnit position="north" header="#{bundle.AppName}">

                <h:graphicImage value="/resources/images/book-header.jpg" alt="logo" />

                <h:form id="menuForm">
                    <p:menubar>
                        <p:menuitem value="#{bundle.Home}" outcome="/admin/index" icon="ui-icon-
home"/>
                        <p:menuitem value="#{bundle.News}" outcome="/admin/news/List" icon="ui-icon-
script"/>
                        <p:menuitem value="#{bundle.Options}" outcome="/admin/Options" icon="ui-icon-
gear"/>
                        <p:menuitem value="#{bundle.Users}" outcome="/admin/person/List" icon="ui-
icon-person"/>
                        <f:facet name="options">
                            <p:commandButton ajax="false" value="#{bundle.Logout}"
action="#{loggedUserController.logout()}" icon="ui-icon-extlink" style="float:right; margin-left:
10px; margin-top: 0px !important;"/>
                            <p:menuitem value="#{bundle.Language}" style="display: table-row;">
                                <p:outputLabel for="languageSelector" value="#{bundle.Language}"
style="display: table-cell; vertical-align: middle; padding-right: 3px"/>
                                <p:selectOneMenu id="languageSelector"
value="#{languageSwitcherBean.locale}" onchange="submit()"

valueChangeListener="#{languageSwitcherBean.localeChanged}" style="display: table-cell; vertical-
align: middle;">
                                    <f:selectItems value="#{languageSwitcherBean.countries}" />
                                </p:selectOneMenu>
                            </p:menuitem>

                            <p:menuitem value="#{bundle.Theme}" style="display: table-row; ">
                                <p:outputLabel for="themeSelector" value="#{bundle.Theme}"
style="display: table-cell; vertical-align: middle; padding-left: 10px; padding-right: 3px"/>
                                <p:selectOneMenu id="themeSelector" style="display: table-cell;
vertical-align: middle;"
                                                 value="#{themeSwitcherBean.pickedTheme}"
                                                 var="theme" effect="drop"
                                                 onchange="$('#menuForm').submit()">
                                    <f:converter converterId="jsf.util.ThemeConverter"/>
                                    <f:selectItems var="t"
                                                   value="#{themeSwitcherBean.themes}"
                                                   itemLabel="#{t.name}"
                                                   itemValue="#{t}"/>
                                    <p:column>
                                        <h:outputText styleClass="ui-theme ui-theme-#{theme.name}" />
                                    </p:column>
                                    <p:column>
                                        <h:outputText value="#{theme.name}"/>
                                    </p:column>
                                </p:selectOneMenu>
```

```
                        </p:menuitem>
                    </f:facet>
                </p:menubar>
            </h:form>
        </p:layoutUnit>
        <p:layoutUnit position="center">
            <center>
                <ui:insert name="body"/>
            </center>
        </p:layoutUnit>
    </p:layout>
</h:body>
</html>
```

g) Wyszukiwanie

Wyszukiwanie jest zrealizowane na tej samej zasadzie co w projekcie I.

h) Przykład wykorzystania biblioteki PrimeFaces

Dzięki tej bibliotece zaimplementowane zostało m.in. wyświetlanie kategorii w postaci drzewa:

```
<p:tree value="#{categoryController.root}" var="node" dynamic="true" id="categoryTree">
        <p:treeNode>
          <h:outputText value="#{node.categoryName}" rendered="#{!categoryController.isList(node)}"/>

          <p:dataList value="#{node}" var="isbn" rendered="#{categoryController.isList(node)}" type="none" paginator="false" >
            <table style="padding:0; margin:0; border:0;border-collapse: collapse; border-spacing: 0;">
              <tr style="padding:0; margin:0; border:0;border-collapse: collapse; border-spacing: 0;">
                <td style="padding:0; margin:0; border:0;border-collapse: collapse; border-spacing: 0;">
                  <p:commandLink update=":IsbnViewForm" oncomplete="PF('IsbnViewDialog').show()" title="#{bundle.ViewDetail}"
styleClass="ui-icon ui-icon-search" style="display:inline-block" value="#{isbn.title}">
                    <h:outputText value="#{isbn.title}" />
                    <f:setPropertyActionListener value="#{isbn}" target="#{isbnController.selected}" />
                  </p:commandLink>
                  <h:outputText value="#{isbn.title}" style="display:inline-block"/>
                </td>
              </tr>
            </table>
          </p:dataList>

        </p:treeNode>
      </p:tree>
```

i) Obsługa skórek:

W projekcie skorzystano ze skórek PrimeFaces. Potrzebne do tego było zadeklarowanie ustawień obecnej skórki w pliku web.xml oraz aktualizacja wyboru poprzez Managed Beana.

```
<context-param>
  <param-name>primefaces.THEME</param-name>
  <param-value>#{themeSwitcherBean.pickedTheme}</param-value>
</context-param>
```

j) Inne:

- W projekcie wykorzystano open source'ową bibliotekę komponentów (framework) PrimeFaces.