Pokročilá laboratoř Java EE

labE-shop

prochka6@fel.cvut.cz

Martin Tomášek

Kamil Procházka

Ondřej Harcuba

Michal Horák

1. Vize a požadavky

Cílem je vytvořit aplikaci určenou k demonstraci použití technologie JAVA EE. Tomuto zadání odpovídá i rozsah práce a funkčnosti, které budou na aplikaci kladeny.

1.1. Funkční požadavky

1. Aplikace bude umět registrovat uživatele.
2. Aplikace bude umožňovat uživateli měnit svá data.
3. Aplikace bude umožňovat zobrazování produktů všem uživatelům.
4. Aplikace bude umožňovat vyhledávání mezi produkty.
5. Aplikace bude umožňovat přidávání a editování produktů.
6. Aplikace bude umožňovat přidat produkty do kategorií.
7. Aplikace bude umět přidávat produkty do košíku pro registrované uživatele.
8. Aplikace bude umět editovat košík.
9. Aplikace bude umožňovat správu uživatelé.
10. Aplikace bude umožňovat správu objednávek.

1.2. Nefunkční požadavky

1. Aplikace bude nasazena na Openshift.
2. Aplikace bude zabezpečena.
3. Aplikace poběží v clusteru.

2. Návrh a implementace

Cílem bylo navrhnout zadanou aplikaci. Na view bylo použito JSF podle zadání. EJB vrstvu jsme vložili do WAR modulu stejně jako DAO vrstvu.

2.1. View

Při implementaci view jsme použili funkce poskytované v JSF, dále jsme se snažili minimalizovat velikost session, takže je mnoho věcí navrženo jako @RequestScope, abychom mohli přistupovat k jednotlivým Beans pomocí expression language je použita anotace @Named, které nám dává přístup k beaně s konkrétním názvem.

2.2. EJB

Aplikace využívá pouze @Stateless EJB beany, které jsme označili jako manažery. Přístup k nim je pomocí CDI a anotace @Inject. Tito manažeři jsou zabezpečeny frameworkem JAAS, více v sekci zabezpečení. Každý manažer je zodpovědný za práci s určitou business logikou a k tomuto účelu může využívat služeb dalších manažerů.

2.3. DAO

Při návrhu datového modelu jsme vycházeli z toho, že systém bude určen k prodávání věcí. Produkty jsou proto součástí určitých kategorií a patří různým výrobcům. Registrovaný uživatel má vlastní košík, do kterého si může vkládat produkty. Košík je ukládán do DB, a proto přežije i pád aplikace. Po vytvoření

objednávky jsou věci z košíku vyhozeny a přetransformovány na objekty, které jsou spárovány s objednávkou. Zde se objevují první duplicitní parametry na cenu, které jsou ovšem v pořádku, neboť předpokládáme, že se ceny produktů mohou měnit.

2.4. Framework a chování aplikace

2.4.1. Maven

- neobsahuje definice repositářů odkud se mají závislosti dotahovat, je potřeba dodat, případně mít vlastni soubor v lokálním repositáři settings.xml

- obsahuje plodin "maven-processor-plugin", který generuje JPA metamodel pro o anotované třídy v projektu, které jsou používaný v Criteria Queries. Spouštíme ho manuálně přikázem mvn generate-sources.

Definice v pom.xml je dostatečné samo popisná. Vygenerovány metamodel se sám přidá na build path. V eclipse si ho tam vsak musíme přidat sami. Vygenerované třídy jsou ve složce target/ generated-sources / apt.

2.4.2. Seam

Využili jsme integrační framework Seam 3. Hlavně z důvodu managovani persistentního kontextu, abychom se vyhli lazy initialization exception a měli automaticky spravovaný persistentní kontext.

Definice ve třidě util.Resources:

```
@SuppressWarnings("unused")

@Produces

@ExtensionManaged

@ConversationScoped

@PersistenceUnit(unitName = "lab-eshop-PU")

private EntityManagerFactory entityManagerFactory;
```

definuje, že o persistentní kontext se stara Seam (anotace @ExtensionManaged), že persistentni kontext bude žit po dobu konverzace. Tato definice nám umožňuje dávat @Inject EntityManager em; to nám vrátí persistentní kontext spravovaný seamema splňujeme podmínku, že zdroje máme typové bezpečné.

U třídy resources je definovaný @Producer pro Loggery používané v aplikaci. Z parametru metody invocationContextu se získá odkud chceme Logger a podle toho vytvoříme ve factory Logger.

```
@Produces
public Logger produceLog(InjectionPoint ip) {
    return LoggerFactory.getLogger(ip.getMember().getDeclaringClass());
}
```

Další možností co nám Seam dává, jsou třeba @Produceri pro běžný request, response objekty @Inject FacesContext context; @Inject HttpServletRequest request; a další. Další vlastnost co je zajímavá je možnost generování FacesMessage zpráv pres interface @Inject Messages message; které jsou definovaném v takzvaném RenderScopu a přežiji i redirect na jinou stránku.

2.4.3. Bug fixy

Abychom vynutili UTF-8 kódovaní implementovali jsme jednoduchý javax.servlet.Filter, který pokud jeho init-param forceEncoding definovatelný ve web.xml je nastaven na true(default), každé request i response na server označí za kódovaný v UTF-8. Tímto způsobem jsme opravili spatně kódovaní přijmutích dat z formulářů z JSF stránek, kdy browser nepřidal charset do HTTP hlaviček.

Pokud formulář (product management) odesílá data ve tvaru multipart/form-data nedodržuji browsery kontrakt o nastavení kódovaní jednotlivých parametru a implementace primefaces neuměla překódovat přijata data. Třídy v balíčku web.util jsou, až na jednořádkovou změnu vynucující překódovaní Stringu z ISO-8859-1 na UTF-8.

Filter předpokládá Servlet 3 prostředí a využívá anotace pro zaregistrování do startu aplikace.

2.4.4. ImageServlet

Naimplementovali jsme servlet, který nám vrací přímo obrázky produktu.

Pro jednoduchost, a protože nemáme, MIME informace o obrazcích předpokládám, že se vždy jedna o image/jpg.

Servlet předpokládá opět Servlet 3 prostředí a registruje se automaticky anotaci na adresu: /dynamic-resources/product/*, kde * na konci reprezentuje ID produktu.

Servlet má nastavitelný <context-param> ve web. xml velikost bufferu pro streamovani obsahu.

Další funkcí co nabízí je možnost zapnuti cachovani pomocí HTTP hlaviček. Samozřejmosti je možnost specifikovat cache expiry seconds.

Defaultne ImageServlet cachuje a to po dobu 5 minut.

Balicek web.producers

obsahuje dve třídy, které zpřístupňuji pomoci anotace @Producer a @Named list kategorii a značek. Navíc co vás může zaskočit je ze obsahuji definici @RequestScope. Ta říká, že pokud někdo chce hodnotu z této metody, tak již není v rámci jednoho REQUESTU znova volaná.

Hodnota vrácena touto metodou je uložena v danem scopu. A pokud by proběhlo několik volání @Inject List<Category> metoda se zavolá pouze jednou, znova se vrátí hodnota cachovana v request scopu CDI frameworkem.

Důvodem tohoto je optimalize, protože volaní get categorie a brands může byt více. Typické selecty a menu mohou volat tuto metodu a data předpokládáme víceméně statická.

2.4.5. DAO

Implementovali jsme ho přes obecný typ, který je ale značně JPA specifický. Obsahuje metody pro flush() a clear(), či mutaci EntityManagera. Není implementován jako EJB. Předpokládá vždy, že nad ním leží nějaká transakci střední vrstva a pouze obaluje všechny databázové dotazy, aby nebyli roztroušené po aplikaci.

```
@Inject

@Override

public void setEntityManager(EntityManager entityManager) {

        this.em = entityManager;

}
```

Díky tomuto kusu kódu vždy když dáme v manažerovi @Inject IXXXDao dao; ma tento DAO objekt inicializovaného EntityManagera. Ale pokud bychom chtěli jiného, máme zde metodu setEntityManager(), tím můžeme sami instanciovat Dao objekty i z jiných míst.

Generická implementace nám umožnila snadné vytváření CRUD Dao objektu.

V rámci DAO vrstvy používáme bud Type Safe, díky metamodelu JPA Criteria Queries, nebo @NamedQueries, která sice nejsou typová, ale stále jsou staticky kompilovaná a server by nás v případě chyby v JPQL upozornil již při nasazovaní aplikace a ne až za běhu, takže se jedna o bezpečně typované dotazy. Navíc ma tu výhodu, že jsou to cachovane prepared statementy, pokud tuto moznost máme povolenou v aplikačním serveru a vejde se do cache. Tím se zrychluje vykonávaní

těchto queries. Všechny queries jsou ošetřené proti SQL injektor a to pomoci parametru.

2.4.6. FILTER

Obsahuje jednoduchou definici filtru, které mohou klienti podědit a DAO vrstva na základě parametru v nich generuje specifické dotazy pomoci Criteria Queries.

Povšimněte si metod JpaUserDao.find () countByFilter(), které slouží User managemntu pro lazy loading uživatelů do dynamické tabulky. Je zde implementace Criteria Queries pro filtrovaní uživatelů.

Sem také patří ProductSearchFilter, který umí filtrovat produkty podle ceny min - max., radit podle ceny, či defaultně titulku asc/desc, kategorie a značky do kterých patří.

U dotazovaní nad produkty jedna důležitá věc.

Kazdy dotaz co se ptá ze zákaznické strany pohledu na produkty děla podmínku, ze PUBLISH DATE <= NOW a DISCARD DATE > NOW.

Tudíž produkty co jsme ještě nepublikovali, anebo již nejsou k prodeji nejsou vidět ve filtrech.

Avšak pokud známe id, či máme stávající objednávku, stále se můžeme přesměrovat na detail produktu.

3. Zabezpečení

Aplikace byla zabezpečena použitím frameworku JAAS. Rozeznáváme celkem 3 druhy uživatelů:

- Guest
- Customer
- Administrator

S tím, že Guest může pouze procházet stránky, avšak nemůže si nic pořídit, nemá k dispozici košík.

Customer je role zákazníka, který může procházet kategorie a vyhledávat produkty, obdobně jako Guest, ale navíc si může produkty zakoupit a prohlížet si historii svých objednávek.

Administrátor je role správce systému, který může vkládat a editovat produkty, kategorie, značky, editovat či mazat objednávky.

Jeden přihlášený uživatel může být v roli jak Customer tak Administrátora, záleží na nastavení. Ovládací prvky ve view jsou podmíněně vykreslovány v závislosti na přihlášeném uživateli a jeho roli. Přístupy do určitých sekcích jsou nastaveny ve web.xml.

Další úroveň zabezpečení je na vrstvě EJB. Volání jednotlivých metod je zabezpečeno pomocí anotací, které specifikují role uživatelů, kteří mohou volat určité funkce.

4. OpenShift

Aplikace běží v cloudovém řešením OpenShift od firmy RedHat. Aplikace je dostupná na adrese: https://eshop-ajeelab.rhcloud.com/

Pro běh eshopu na OpenShiftu bylo nutné vytvořit několik catridges:

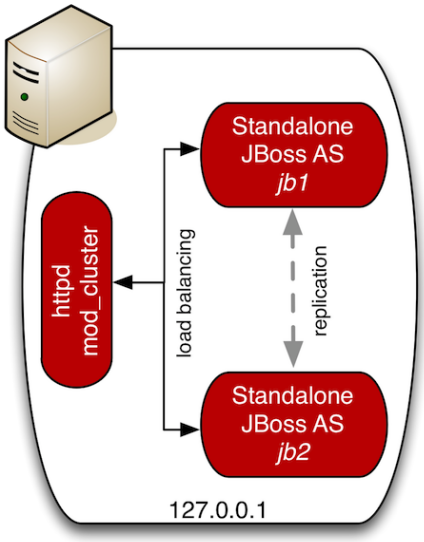
- JBoss Application Server 7.1
- MySQL Database 5.1
- phpMyAdmin 3.4 (testování)

Během vývoje nenastaly žádné problémy, kvůli kterým by nebylo možné aplikaci na OpenShiftu nasadit.

5. Clustering

Aby byla aplikace splňovala kriteria pro vysokou dostupnost (high availability), byla připravena a nakonfigurována pro nasazení do více clusterů.

Pro naše potřeby a testování jsme eshop nasadili do clusteru dvou uzlů, které jsou na jednom počítači, na kterém také běží i Apache httpd server s pluginem mod_cluster, který slouží jako load balancer:



Obr1.: Cluster dvou uzlů s load balancerem.
Zdroj: http://akquinetblog.files.wordpress.com

Obě instance serverů jsou spouštěny s předpřipravenou konfigurací pro clustering (standalone-ha.xml). Pro každý uzel bylo v konfiguraci zvoleno jiné jméno. Jeden ze dvou uzlů je spouštěn s nastaveným offsetem, tak aby porty dvou uzlů, které aplikační servery využívají, nebyly stejné.

Všechny náležitosti, které bylo nutné splnit pro to, aby aplikace byla nasazena v clusteru dvou uzlů, byly splněny. Jako ověření toho, že je aplikace nasazena správě, je to, že ve výpisu logů lze dohledat tyto hlášky:

```
...
00:53:46,178 INFO
[org.jboss.as.clustering.impl.CoreGroupCommunicationService.ejb] (MSC service
thread 1-8) JBAS010206: Number of cluster members: 2
00:53:46,178 INFO
[org.jboss.as.clustering.impl.CoreGroupCommunicationService.web] (MSC service
thread 1-6) JBAS010206: Number of cluster members: 2
01:09:57,456 INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-17,null) ISPN000094: Received new cluster view: [standalone-node2/web|3]
[standalone-node2/web, standalone-node1/web]
...
```

Obr 2.: Mod Cluster Manager (http://localhost/mod_cluster_manager)

5.1. Failover (Testování odolnosti vůči výpadku jednoho uzlu)

Odolnost aplikace vůči výpadku jednoho uzlu se nám bohužel nepovedlo demonstrovat. Testování probíhalo následujícím způsobem:

- Spuštění AS 7.1 (node1)
- Spuštění AS 7.1 (node 2)
- Odeslání požadavku na webový server, na kterém je spuštěn load balancer mod_cluster (v našem případě http://localhost/eshop).
- Stránka se zobrazí bez problému.
- Uzel, který požadavek obsluhoval killneme.
- Nyní by mělo dojít k tomu, že při následném obnovení stránky požadavek obslouží druhý uzel.

V naší aplikaci dojde k následující chybě:

```
19:57:42,618 ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host]] (ajp--127.0.0.1-8109-1) Exception sending request initialized lifecycle
event to listener instance of class org.jboss.weld.servlet.WeldListener:
java.lang.IllegalStateException: Error restoring serialized contextual with id
org.jboss.weld.bean-lab-eshop.war/D:/SKOLA/MagisterskeStudium/AJEELab/jboss-as-
7.1.1.Final/standalone/deployments/lab-eshop.war/WEB-INF/classes-ManagedBean-class
cz.cvut.fel.jee.labEshop.web.LoginBean
        at
org.jboss.weld.context.SerializableContextualImpl.loadContextual(SerializableConte
xtualImpl.java:91) [weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
```

```
        at
org.jboss.weld.context.SerializableContextualImpl.get(SerializableContextualImpl.j
ava:79) [weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at
org.jboss.weld.context.SerializableContextualImpl.delegate(SerializableContextualI
mpl.java:37) [weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at
org.jboss.weld.context.ForwardingContextual.toString(ForwardingContextual.java:46)
[weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at java.lang.String.valueOf(String.java:2902) [rt.jar:1.7.0_04]
        at java.lang.StringBuilder.append(StringBuilder.java:128)
[rt.jar:1.7.0_04]
        at
org.jboss.weld.context.SerializableContextualInstanceImpl.toString(SerializableCon
textualInstanceImpl.java:60) [weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at java.lang.String.valueOf(String.java:2902) [rt.jar:1.7.0_04]
        at java.lang.StringBuilder.append(StringBuilder.java:128)
[rt.jar:1.7.0_04]
        at
org.jboss.weld.context.beanstore.AttributeBeanStore.attach(AttributeBeanStore.java
:109) [weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at
org.jboss.weld.context.AbstractBoundContext.activate(AbstractBoundContext.java:66)
[weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at
org.jboss.weld.servlet.WeldListener.requestInitialized(WeldListener.java:141)
[weld-core-1.1.5.AS71.Final.jar:2012-02-10 15:31]
        at
org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:143)
[jbossweb-7.0.13.Final.jar:]
        at
org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:102)
[jbossweb-7.0.13.Final.jar:]
        at
org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:109)
[jbossweb-7.0.13.Final.jar:]
        at
org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:368)
[jbossweb-7.0.13.Final.jar:]
        at org.apache.coyote.ajp.AjpProcessor.process(AjpProcessor.java:505)
[jbossweb-7.0.13.Final.jar:]
        at
org.apache.coyote.ajp.AjpProtocol$AjpConnectionHandler.process(AjpProtocol.java:44
5) [jbossweb-7.0.13.Final.jar:]
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:930)
[jbossweb-7.0.13.Final.jar:]
        at java.lang.Thread.run(Thread.java:722) [rt.jar:1.7.0_04]
```

Výše uvedenou chybu jsme se snažili odstranit. Chyba naznačuje, že je problém s deserializací nějaké proměnné. Prvním krokem bylo to, že jsme odstranili „teoreticky podezřelé kandidáty“, jako je @Inject Logger log, @Inject UserManager userManager. Ani odstranění všech těchto proměnných nepomohlo. Chyba se objevovala i poté, co ve třídě LoginBean byly zakomentovány všechny proměnné a těla metod.

Stejná chyba nastala v prostředí Windows 7 i Ubuntu (Quantal Quetzal). Chyba se objevovala i na nové verzi JBoss AS 7.1.2

Stejný problém je i na jboss foru, kde však problém není vyřešen. (https://community.jboss.org/thread/195690)