České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Petr Dobřička**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Adaptivní elektronický obchod**

Pokyny pro vypracování:

1. Zjistěte, zda nástroje pro tvorbu portálu elektronického obchodu obsahují nějaké možnosti adaptace prostředí podle zájmů uživatele.
2. Navrhněte prostředí administrační a veřejné části elektronického obchodu.
3. Webovou aplikaci implementujte s využitím frameworků Spring, Hibernate a Primefaces. Jako základ aplikace použijte Adaptive System Framework ASF (dodá vedoucí).
Do aplikace vhodně integrujte adaptivní chování.
4. Výslednou aplikaci zprovozněte na Google AppEngine a popište způsob publikování z vývojového prostředí Netbeans. Aplikaci otestujte a vyhodnoťte adaptivní chování.

Seznam odborné literatury:

[1] Brusilovsky P., Kobsa A., Nejdl W.: The Adaptive Web, Springer, 2007.
[2] Walls C.: Spring in Action, third edition, Manning Publications, 2011.
[3] Sanderson D.: Programming Google App Engine, O’Reilly Media, 2009.

Vedoucí: Ing. Martin Balík

Platnost zadání: do konce zimního semestru 2013/2014

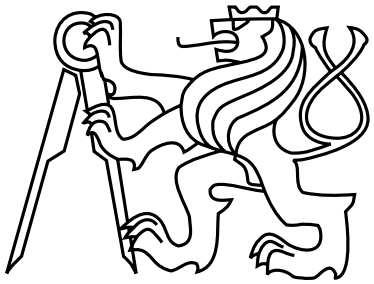
L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 1. 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky



Bakalářská práce

Adaptivní elektronický obchod

Petr Dobřička

Vedoucí práce: Ing. Martin Balík

Studijní program: Kybernetika a robotika, strukturovaný, Bakalářský

Obor: Systémy a řízení

3. ledna 2014

Poděkování

Především bych rád poděkoval Ing. Martinu Balíkovi za trpělivost a ochotu dovést mě ke zdárnému konci. Samozřejmě také všem ostatním, kteří mě podporovali v psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Plzni dne 3. 1. 2014 .....

Abstract

This bachelor thesis deals with the creation of an electronic shop, which includes an adaptive behavior. To implement adaptive behavior is used Adaptive System Framework. The application is developed with NetBeans by Java language and uses frameworks Spring and PrimeFaces. To work with data is used Java Persistence API and the application is adapted to function on the Google App Engine.
The goal of this work is to design and implement the electronic shop, integrate adaptive behavior, upload on Google App Engine and execute testing. The work contains guide to setup aplication for Google App Engine by Apache Maven too.

Abstrakt

Tato bakalářská práce se zabývá tvorbou elektronického obchodu, který obsahuje adaptivní chování. K zavedení adaptivního chování je použit Adaptive System Framework. Aplikace je vyvinuta v NetBeans pomocí jazyku Java a pomocí frameworků Spring a PrimeFaces. Pro práci s daty je použito Java Persistence API a aplikace je přizpůsobena pro fungování na Google App Engine.
Cílem práce je navrhnout a implementovat vlastní obchod, integrovat do něj adaptivní chování, nahrát ho na Google App Engine a provést testování. Práce také obsahuje návod nastavení aplikace pro Google App Engine pomocí Apache Maven.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem této bakalářské práce je vytvoření elektronického obchodu, do kterého bude možné integrovat adaptivní chování a následně ho nahrát na Google App Engine (GAE)[10]. Důvod, proč jsem si vybral toto téma je jednoduchý. Z mého pohledu se elektronické obchody určitě uplatní i v budoucnu a kvalitní adaptace obsahu, se u nich stává samozřejmostí. Další důvodem je práce s GAE, která mě zaujala.

V této práci implementuji základní strukturu obchodu, která využije Adaptive System Framework (ASF)[7]. S pomocí ASF se pokusím vtisknout obchodu základní adaptivní chování. Kvůli odlišné datové struktuře datového úložiště a implementace Java Persistence API (JPA)[8] vznikl v ASF na základě mých úprav JPA modulu nový modul specifický pro GAE. V práci také provedu testování tohoto chování. Celý projekt bude vyvíjen tak, aby mohl běžet na GAE. I když práce s GAE byla pro mě úplná novinka, nečekal jsem, že bude tak těžké se s touto službou vypořádat. Proto se v práci nachází i návod, jak jí nastavit.

1.1 Motivace

V dnešní době internetu jsou elektronické obchody výbornou možností pro provozování živnosti. Téměř každá firma má nyní své webové stránky. Pokud navíc obchoduje s lehce prodejným zbožím přes internet jako jsou knihy, hudba nebo elektronika, zvážení vlastnit internetový obchod, je zcela na místě. Určitě se uživí i obchody s jiným zbožím, ale pro mě osobně je lepší si oblečení nebo boty vyzkoušet než si je koupím. Majitel takto získá přístup k milionům potenciálních zákazníků za téměř nulovou režii. Sám často uvažuji o přivýdělku pomocí takovéhoto obchodu. Pokud vezmu v potaz ceny nástrojů pro tvorbu internetových obchodů a hostingu, protože nepočítám s obchodem pro široké masy, přijde mi jako dobrý nápad si takovýto obchod vytvořit sám a nahrát ho na bezplatný hosting, který nabízí GAE.

Protože zákazník většinou předem neví, co přesně za produkt chce, přichází na scénu adaptace. Adaptivní chování má usnadnit zákazníkovi výběr podle jeho osobních preferencí. Pokud si zákazník vybere rychle a dobře, tak bude pravděpodobně spokojen a je tu velká šance, že obchod použije znovu, případně ho doporučí svému okolí. Z toho samozřejmě těží majitel obchodu a jeho snahou je, aby se jeho zboží prodávalo jednoduše na pár kliknutí. Jak je vidět v následující kapitole 2, tak v dnešní době skoro každý obchod obsahuje adaptační část, proto by neměla chybět ani v mém obchodě.

Kapitola 2

Teoretická část

2.1 Nástroje pro tvorbu obchodů

V zadání stojí, abych zjistil možnosti adaptace u současných nástrojů na tvorbu elektronických obchodů. Našel jsem několik stránek, které se zabývají e-commerce a obsahují seznamy nejlépe hodnocených nástrojů pro tvorbu obchodů. Vybral jsem několik nástrojů, které mají nejlepší hodnocení a prozkoumal jejich dokumentaci a možnosti.

Nejlepší hodnocení má Volusion. Prošel jsem možnosti práce s produkty a správou uživatelů, které Volusion nabízejí. Je tu možnost zobrazení příbuzných produktů, ale zmiňují, že tento výběr je náhodný z dané kategorie. Pokud uživatel chce výběr ovlivnit, jediná možnost je nastavení doporučeného produktu ručně. Nástroj také obsahuje hodně možností pro filtrování a možnost psát uživatelské recenze k produktům.

Další dobře hodnocený nástroj je Bigcommerce. Zde, kromě tradičních možností pro filtrování, recenze a hodnocení produktů, je také jejich speciální engine, který již zajišťuje adaptaci. Konkrétně navrhuje zákazníkům srovnatelné produkty vybrané na základě jejich nákupních položek. Také obsahuje porovnání produktů.

Další nástroj je Shopify. Na první pohled dobře vypadající a jednoduchý nástroj na tvorbu plně funkčního online obchodu i s hostingem. Z dokumentace je patrné, že nabízí širokou škálu možností. Mezi nimi je 5 způsobů jak doporučovat zákazníkům zboží. Každý ze způsobů má své specifické vlastnosti, které jdou i kombinovat.

Ashop commerce nástroj. Po přístupu na jejich stránky, mi byl nabídnut online chat se zaměstnancem, takže jsem přeskočil procházení dokumentace a rovnou se zeptal. Z odkazu, co mi dotyčný předal, jsem vyčetl, že mají feature, která automaticky nabízí 1-4 podobné produkty. Bohužel uživatel si ji nemůže nějak přesněji nastavit, jen počet nabízených podobných produktů. Podrobnější dokumentaci této feature jsem ve veřejně přístupné části nenašel.

Poslední nástroj, který zde uvedu, je Pinnacle Cart. Ten také podporuje velké množství možností. Z adaptace nabízí doporučené produkty několika způsoby. Také má u potvrzení objednávky oblasti s "ostatní koupili také".

Odstrašujícím příkladem je web.com. Ani po notné dávce času, kterou jsem na jejich stránkách v části e-commerce strávil, se mi nepodařilo najít jakýkoliv odkaz na nějakou dokumentaci. Seznam nabízených features mi v 6 bodech sděloval pouze základní informace.

Pro jakýkoliv další krok nebo detail je nutná registrace, která zákazníka nutí přímo si vybrat jeden z jejich balíčků nebo se dotázat firmy telefonicky.

Název nástroje	**Domovská stránka**	**Adaptace**
Volusion	www.volusion.com	Částečná
Bigcommerce	www.bigcommerce.com	Ano
Shopify	www.shopify.com	Ano
Ashop Commerce	www.ashopcommerce.com	Ano
Pinnacle Cart	www.pinnaclecart.com	Ano
web.com	www.web.com/ecommerce/	Neurčeno

Tabulka 2.1: Přehled nástrojů k tvorbě e-shopů

Jak je vidět, každý z dobře hodnocených nástrojů, má alespoň nějakou část, která se přizpůsobuje, viz Tabulka 2.1. Odhaduji, že většina se spíše zaměřuje na produkty, ale minimálně Pinnacle Cart a Bigcommmerce sledují i uživatele.

2.2 Google App Engine

V této části bych rád popsal, co to vlastně GAE je [10]. Jedná se o službu, která zajišťuje hosting webových aplikací na serverech společnosti Google. Na GAE se stále pracuje, vyvíjí se. Nyní podporuje aplikace psané v programovacím jazyku Java nebo Python a nově, experimentálně i v PHP nebo Go. Výhoda GAE spočívá ve správě zdrojů, které jsou aplikaci dostupné. Pokud máme náročnou aplikaci a využívá jí více uživatelů, GAE jí automaticky přidělí více zdrojů, aby zabránil poklesu výkonu. Do určité míry je také bezplatný. Pokud nám limity nestačí, je možnost nastavit placení a GAE si bude účtovat[1] jen za zdroje, které aplikace skutečně využívá. GAE Java aplikace nyní běží na Java 7 JVM v sandbox prostředí. K vývoji Java aplikací pro GAE se používá App Engine Java Software Development Kit (SDK)[2]. Nedávno GAE změnil SDK z Java 6 na 7, sice ještě stále podporuje Java runtime 6, ale i to se v budoucnu bude měnit[2]. Další výhodou je, že se nemusíme starat o správu a nastavení serveru a s tím související problémy. Na druhou stranu běh v sandbox modu nás může limitovat. Uvedu zde pár příkladů, jak se GAE chová:

- Modifikuje, přidává a odebírá některé části hlavičky dotazů a odpovědí.
- Pokud se nesplní podmínka časového limitu odpovědi, který je nastaven na 60 sekund, server automaticky vrátí odpověď HTTP 500 server error.
- Nejde zapisovat soubory do systému, pouze do datastore, ale lze je číst a všechny nahrané aplikační soubory jsou dostupné.
- Nemohou se používat systémová volání.
- Vlákna, která mohou přežít dotaz, jdou vytvářet pouze v backend instancích.

[1] Ceník GAE lze najít na https://developers.google.com/appengine/pricing

GAE může pracovat se třemi druhy datových úložišť. Je to Google Cloud SQL[2], Google Cloud Storage[3] a App Engine Datastore. První dva jsou pro větší aplikace a jsou placené, proto použiji Datastore. Datastore poskytuje rozloženou NoSQL službu, která představuje query engine a transakce. Datastore podporuje dvě klasická rozhraní Java Data Object a Java Persistence API. Jejich implementaci zajišťuje DataNucleus Access Platform. Datastore rozděluje vytvořené objekty (entity) do skupin, které se nazývají "entity groups". Všechny entity, které mají stejného předka, jsou v jedné "entity group"a uloženy v datastore pohromadě kvůli lepšímu výkonu. Transakce pracuje s entitami pouze z jediné skupiny. Pokud zapneme Cross Group Transactions, tak může pracovat až na 5 skupinách.

2.3 Adaptivní webový systém

Pod pojmem adaptivní webový systém, si lze představit internetovou aplikaci, kde každý uživatel má svůj model (User Model [9]) a podle jeho obsahu systém přizpůsobuje své reakce. To umožňuje, aby na uživatele s různými profily reagoval rozdílně. Je to značně rozsáhlé téma, proto zde uvedu pouze základní informace a více se budu věnovat návrhu konkrétní adaptace pro můj obchod.

Pro adaptaci je tedy zásadní, správně namodelovat uživatele. Hlavní vlastnosti, které se v dnešní době modelují, jsou znalosti, zájmy, cíle, pozadí, individuální schopnosti a kontext práce. Každý systém používá nějakou množinu z těchto vlastností k modelování uživatele. Vzdělávací programy se zaměřují na znalosti, zatímco třeba informační a doporučující systémy, se zaměřují spíše na zájmy. Nejrozšířenější přístup k modelování uživatelů je nyní overlay user modeling [9]. Pro vytvoření profilu uživatele je potřeba získat o něm nějaké informace. Jsou dvě cesty, jak toho docílit. Explicitně, pomocí komunikace s uživatelem a nechat ho informace zadat. K tomu mohou sloužit různá nastavení profilu, tlačítka líbí/nelíbí, formuláře a další. A implicitně, pomocí pozorování chování uživatele. První zmíněná cesta zvyšuje zátěž na uživatele, což některé může obtěžovat, ale zátěž zvyšuje i instalace softwarového agenta, který bude uživatele pozorovat. Jedna z možností reprezentace profilu, je pomocí klíčových slov. Spočívá v setu výrazů, kde každý má svoji spočítanou váhu v daném profilu. Další je sémantická síť pojmů, kde každý uzel představuje pojem a větve spojují související pojmy. Podobně jako sémantická síť funguje i pojmový profil. Zde ale uzly tvoří abstraktní témata, která by mohla zajímat uživatele. Profil lze vytvořit ručně, ale to je časově náročné, proto se preferuje automatická tvorba a údržba.

Samotná adaptace se dělí do dvou hlavních skupin. První je adaptace obsahu, kterou lze rozdělit na adaptaci textu a adaptaci multimedií. Druhá je adaptace odkazů, do které patří skrývání odkazů, tvorba nových, třídění, přímé navádění a další. Nejjednodušší je přímé navádění, které dle uživatelského profilu nabídne další nejlepší odkaz a pokud stránka již odkaz obsahuje, tak ho zvýrazní.

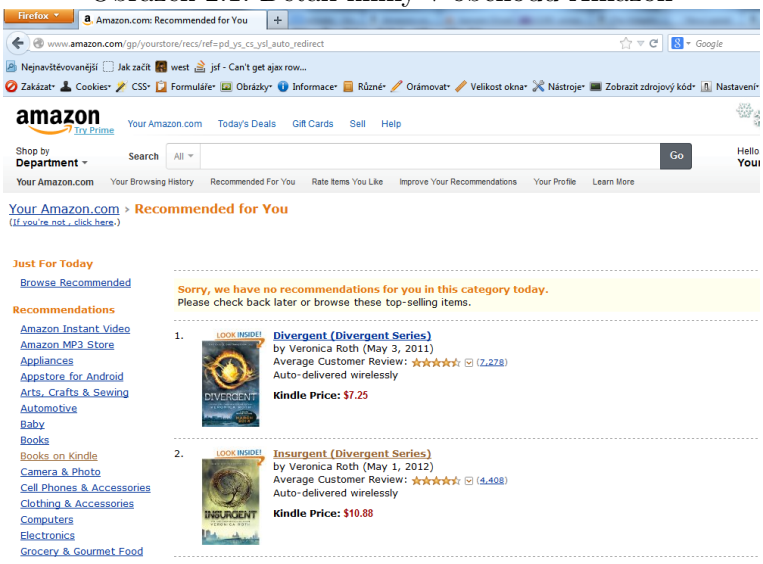
Mezi adaptivní webové systémy se řadí i obsahově založené doporučovací systémy. A mezi ně patří elektronické obchody. Pracuje se s popisky produktů, ty často obsahují nestrukturovaný text, proto je třeba upravit určování váhy jednotlivých klíčových slov. Jedna z metod,

[2] https://developers.google.com/cloud-sql/
[3] https://developers.google.com/storage/

kterou lze použít je TF-IDF [9]. K vyhodnocení pravděpodobných odpovídajících předmětů, lze použít například metodu nejbližšího souseda.

Ukážu zde dva případy adaptace z adaptivního webu Amazon[4]. První, je adaptace podle aktuálního zobrazeného produktu. Na stránce detailu produktu se zobrazí, jaké další knihy si také zobrazili ostatní uživatelé, kteří navštívili tento produkt, Obrázek 2.1. Druhý případ je nabídnutí doporučených produktů přihlášeným uživatelům. Já jsem si na tomto serveru zatím žádnou knihu nekoupil a ani jsem si nenastavil profil. Z toho důvodu se pravděpodobně nevytvořil plnohodnotný uživatelský profil a adaptace mi nedoporučí nic, alespoň mi nabídne seznam nejprodávanějších produktů dané kategorie, Obrázek 2.2.

Obrázek 2.1: Detail knihy v obchodu Amazon



Obrázek 2.2: Doporučené produkty pro uživatele v obchodu Amazon

[4]http://www.amazon.com/

Kapitola 3

Návrh aplikace

V této kapitole se pokusím navrhnout obchod. Využiji k tomu Unified Modeling Language (UML)[6]. Začnu vymezením požadavků, poté navrhnu případy užití, strukturu obchodu a uživatelské rozhraní.

3.1 Požadavky

Ze zadání práce je patrné, že mám vytvořit implementaci elektronického obchodu, jejíž součástí bude adaptace obsahu. K tomu mám použít vývojové prostředí NetBeans a programovací jazyk Java. Navrhnu kompletní obchod, ale implementovat budu pouze části potřebné pro správnou funkci a testování adaptace. K zavedení adaptivního chování do aplikace budu používat framework, který mi dodal a vyvíjí vedoucí práce Ing. M. Balík, Adaptive System Framework[7] . Další z požadavků je, aby aplikace fungovala na Google App Engine[10]. V aplikaci bych měl dle zadání využít také frameworky Spring[11] a PrimeFaces[5]. Pro přístup k datům použiji Java Persistence API[8]. Z toho vyplývají následující funkční a nefunkční požadavky.

3.1.1 Funkční požadavky

- Obchod bude mít administrační a veřejnou část.
- Administrační část bude přístupná pouze přihlášeným zaměstnancům.
- Veřejná část bude přístupná i nepřihlášeným zákazníkům.
- Veřejná část bude obsahovat zboží rozdělené do kategorií.
- Detail zboží bude obsahovat adaptované doporučené zboží.
- Zákazník bude moci procházet nabízené zboží a spravovat svůj košík.
- Pouze přihlášený zákazník bude moci z košíku udělat objednávku.
- Zboží bude moci být hodnoceno pouze přihlášenými zákazníky.

3.1.2 Nefunkční požadavky

- Použití vývojového prostředí NetBeans.
- Použití jazyku Java.
- Použití adaptačního frameworku ASF.
- Nasazení na GAE.
- Použití frameworků Spring a PrimeFaces.
- Přístup k datům pomocí JPA.

3.2 Případy užití

Protože celý elektronický obchod společně s adaptací v jednom diagramu by byl složitější, nepřehledný a příliš velký, rozdělím ho na veřejnou a administrační část. Všechny dialogy uživatel může ukončit a v tom případě se nic nezmění. Pokud se objeví nějaká chyba, je o ní uživatel informován.

3.2.1 Veřejná část - případy užití

Diagram případů užití je na Obrázku 3.1.

3.2.1.1 Zobrazit seznam produktů dané kategorie

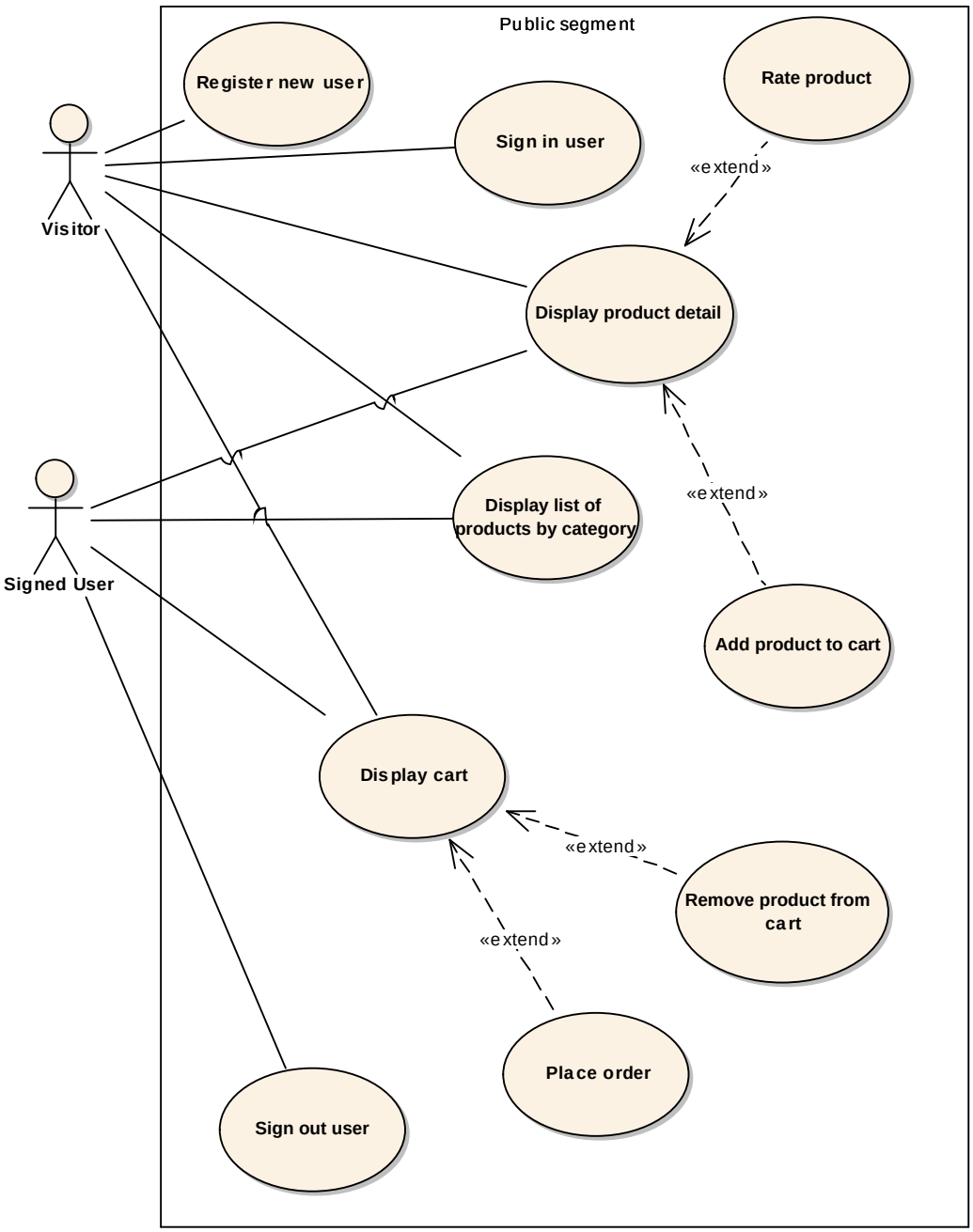
Popis: Zobrazí stránku s tabulkou produktů z vybrané kategorie.
Aktéři: Návštěvník, přihlášený uživatel
Scénář:

1. Uživatel klikne na kategorii, která obsahuje produkty.
2. Systém načte náhledy produktů.
3. Náhledy se uspořádají do tabulky, která je poté prezentována uživateli.

3.2.1.2 Zobrazit detail produktu

Popis: Zobrazí stránku s detailem produktu spolu s doporučenými produkty.
Aktéři: Návštěvník, přihlášený uživatel
Scénář:

1. Uživatel klikne na náhled produktu v tabulce kategorie z případu 3.2.1.1 nebo klikne na doporučený produkt.
2. Systém načte produkt.
3. Systém načte doporučené produkty na základě adaptace (Obrázek 3.4).
4. Uživateli je zobrazen detail produktu spolu s doporučenými produkty, které jsou schopné spustit případ 3.2.1.2 .



Obrázek 3.1: Případ užití - administrační část

3.2.1.3 Přidání produktu do košíku

Popis: Přidá položku s daným produktem a jeho počtem do košíku uživatele.
Aktéři: Návštěvník, přihlášený uživatel
Scénář:

1. Uživatel klikne na tlačítko přidat do košíku v případě 3.2.1.2.
2. Systém se zeptá v dialogu zákazníka, kolik produktů chce do košíku přidat.
3. Uživatel vybere počet a potvrdí přidání.
4. Systém uloží položku s produktem a daným počtem do uživatelova košíku.
5. Systém zavře dialog.

3.2.1.4 Hodnocení produktu

Popis: Ohodnotí určitý produkt hodnocením daného uživatele.
Aktéři: Přihlášený uživatel
Scénář:

1. Uživatel klikne na tlačítko ohodnotit produkt v případě 3.2.1.2.
2. Systém se zeptá v dialogu zákazníka jaké hodnocení chce udělit.
3. Uživatel zadá hodnocení a potvrdí dialog.
4. Systém uloží dané hodnocení k produktu i uživateli.
5. Systém zavře dialog.

3.2.1.5 Zobrazení košíku

Popis: Zobrazí obsah košíku daného uživatele v přehledné tabulce.
Aktéři: Návštěvník, přihlášený uživatel
Scénář:

1. Uživatel klikne na tlačítko košíku v záhlaví stránky.
2. Systém načte položky z košíku, náhledy produktů.
3. Systém spočítá celkovou cenu košíku.
4. Uživateli je prezentována tabulka s náhledy produktů, s jejich počtem v košíku a celková cena košíku.

3.2.1.6 Odebrání produktu z košíku

Popis: Odebere vybraný produkt z košíku uživatele.
Aktéři: Návštěvník, přihlášený uživatel
Scénář:

1. Uživatel klikne na produkt v tabulce v případě 3.2.1.5.
2. Systém se zeptá uživatele v dialogu, zda chce skutečně produkt odebrat.
3. Po potvrzení odebere produkt z košíku.
4. Zavolá se případ 3.2.1.5.

3.2.1.7 Podání objednávky

Popis: Vytvoří objednávku ze současného stavu košíku daného uživatele.
Aktéři: Přihlášený uživatel
Scénář:

1. Uživatel klikne na tlačítko objednat v případě 3.2.1.5.
2. Systém se zeptá uživatele na adresu, způsob dopravy, případně způsob platby.
3. Po potvrzení údajů od uživatele vytvoří systém novou objednávku s položkami z košíku a košík vyčistí.
4. Uživateli je zobrazena informace, že objednávka byla úspěšně podána.

3.2.1.8 Registrace nového uživatele

Popis: Vytvoří účet novému uživateli, na který se bude moci přihlašovat.
Aktéři: Návštěvník
Scénář:

1. Uživatel klikne na tlačítko registrovat v záhlaví stránky.
2. Systém se zeptá uživatele na registrační údaje.
3. Po potvrzení údajů od uživatele a kontrole, zda již náhodou neexistuje, vytvoří systém nového uživatele.
4. Uživateli je zobrazena informace, že byl úspěšně registrován.

3.2.1.9 Přihlášení uživatele

Popis: Přihlásí do systému uživatele, pokud je to administrátor, je přesměrován do administrační části.
Aktéři: Návštěvník
Scénář:

1. Uživatel klikne na tlačítko přihlásit se v záhlaví stránky.
2. Systém se zeptá uživatele na přihlašovací údaje.
3. Po potvrzení údajů od uživatele systém ještě zkontroluje práva.
4. Uživatel je na základě práv přesměrován buď na úvodní stránku obchodu nebo do administrační části.

3.2.2 Administrační část - případy užití

Diagram případů užití je na Obrázku 3.2.

3.2.2.1 Zobrazit seznam objednávek

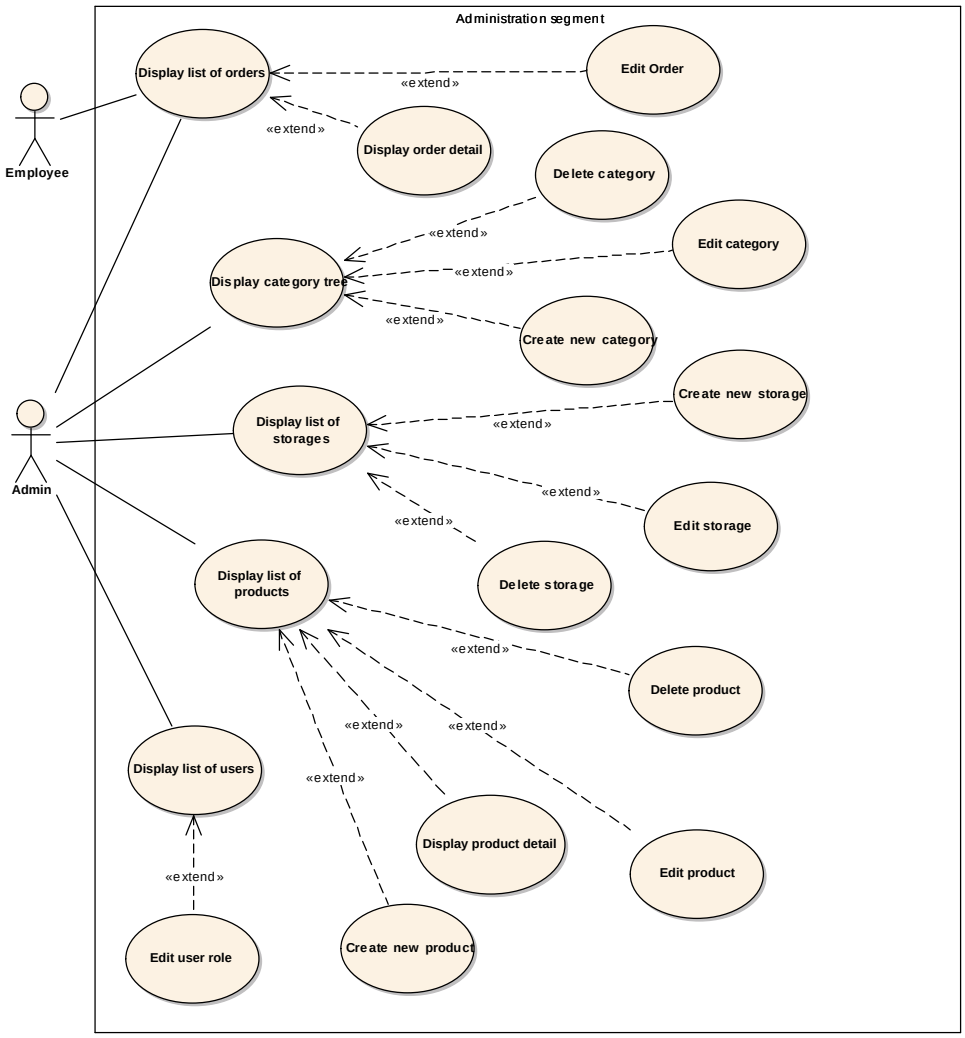
Popis: Zobrazí tabulku objednávek. Jaké objednávky se v tabulce objeví, to záleží na tom, jaký status objednávek si uživatel zvolil.
Aktéři: Administrátor, zaměstnanec
Scénář:

1. Uživatel klikne na jedno z tlačítek v navigaci odkazující na objednávky.
2. Systém určí, jaké tlačítko bylo zvoleno a podle toho načte objednávky se zvoleným statusem do tabulky.
3. Uživateli je zobrazena tabulka s náhledy na objednávky.

3.2.2.2 Zobrazit detail objednávky

Popis: Zobrazí detail objednávky.
Aktéři: Administrátor, zaměstnanec
Scénář:

1. Uživatel klikne na tlačítko detail, pokud má v tabulce z případu 3.2.2.1 vybranou objednávku.
2. Systém načte zvolenou objednávku.
3. Uživateli je zobrazen detail objednávky.



Obrázek 3.2: Případ užití - administrační část

3.2.2.3 Upravit status objednávky

Popis: Změní status objednávky.
Aktéři: Administrátor, zaměstnanec
Scénář:

1. Uživatel klikne na tlačítko upravit, pokud má v tabulce z případu 3.2.2.1 vybranou objednávku.
2. Systém uživateli v dialogu nabídne změnu statusu.
3. Uživatel zadá nový status a potvrdí změnu.
4. Systém změní status objednávky a informuje o uložení změn.

3.2.2.4 Zobrazit seznam produktů

Popis: Zobrazí seznam produktů podle parametrů filtru v tabulce.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko produkty v navigaci nebo na tlačítko hledat u filtru.
2. Systém podle parametrů filtru načte produkty do tabulky.
3. Uživateli je zobrazena tabulka s produkty.

3.2.2.5 Zobrazit detail produktu

Popis: Zobrazí administrátorský detail produktu.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko detail, pokud má v tabulce z případu 3.2.2.4 vybraný produkt.
2. Systém načte zvolený produkt spolu s jeho kategoriemi.
3. Uživateli je zobrazen detail produktu spolu s kategoriemi, ve kterých se nachází.

3.2.2.6 Upravit produkt

Popis: Změní údaje uvedené v produktu, kategorie, obrázky.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko upravit, pokud má v tabulce z případu 3.2.2.4 vybraný produkt.
2. Systém uživateli nabídne stránku s formulářem pro editaci produktu.

3. Uživatel provede změny a klikne na tlačítko potvrdit.
4. Systém zkontroluje uživatelem zadaná data a uloží je.
5. Uživatele informuje o úspěšném uložení změn.

3.2.2.7 Vytvořit nový produkt

Popis: Vytvoří nový produkt v obchodě.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko nový u případu 3.2.2.4.
2. Systém uživateli nabídne stránku s formulářem pro tvorbu nového produktu.
3. Uživatel zadá data, případně nahraje obrázek a klikne na tlačítko vytvořit.
4. Systém zkontroluje uživatelem zadaná data a vytvoří nový produkt, případně obrázky.
5. Uživatele informuje o úspěšném vytvoření nového produktu.

3.2.2.8 Smazat produkt

Popis: Odebere produkt z obchodu.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko smazat, pokud má v tabulce z případu 3.2.2.4 vybraný produkt.
2. Systém uživateli nabídne potvrzovací dialog.
3. Uživatel potvrdí smazání.
4. Systém zkontroluje, zda jde produkt smazat a smaže ho.
5. Uživatele informuje o úspěšném smazání.

3.2.2.9 Zobrazit strom kategorií

Popis: Zobrazí kategorie uspořádané do stromu.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko kategorie v navigaci.
2. Systém načte kategorie do stromu.
3. Uživateli je zobrazen strom s kategoriemi.

3.2.2.10 Upravit kategorii

Popis: Změní jméno případně rodiče kategorie.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko upravit, pokud má ve stromu z případu 3.2.2.9 vybranou kategorii.
2. Systém uživateli nabídne dialog s formulářem pro editaci kategorie.
3. Uživatel provede změny a klikne na tlačítko potvrdit.
4. Systém zkontroluje uživatelem zadaná data a uloží je.
5. Uživatele informuje o úspěšném uložení změn.

3.2.2.11 Vytvořit novou kategorii

Popis: Vytvoří novou kategorii v obchodě.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko nový u případu 3.2.2.9.
2. Systém uživateli nabídne dialog s inputem pro zadání jména a rodiče kategorie.
3. Uživatel zadá jméno a vybere rodiče a klikne na tlačítko vytvořit.
4. Systém zkontroluje uživatelem zadané jméno a vytvoří novou kategorii s vybraným rodičem.
5. Uživatele informuje o úspěšném vytvoření nové kategorie.

3.2.2.12 Smazat kategorii

Popis: Odebere kategorii z obchodu.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko smazat, pokud má ve stromě z případu 3.2.2.9 vybranou kategorii bez potomků.
2. Systém uživateli nabídne potvrzovací dialog.
3. Uživatel potvrdí smazání.
4. Systém zkontroluje, zda kategorii lze smazat a smaže ji.
5. Uživatele informuje o úspěšném smazání.

3.2.2.13 Zobrazit seznam skladovacích míst

Popis: Zobrazí skladovací místa v tabulce.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko skladovací místa v navigaci.
2. Systém načte skladovací místa do tabulky.
3. Uživateli je zobrazena tabulka se skladovacími místy.

3.2.2.14 Upravit skladovací místo

Popis: Přejmenuje skladovací místo.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko upravit, pokud má v tabulce z případu 3.2.2.13 vybrané skladovací místo.
2. Systém uživateli nabídne dialog s inputem pro nové jméno.
3. Uživatel zadá nové jméno a klikne na tlačítko potvrdit.
4. Systém zkontroluje uživatelem zadané jméno a uloží upravené skladovací místo.
5. Uživatele informuje o úspěšném uložení změn.

3.2.2.15 Vytvořit nové skladovací místo

Popis: Vytvoří nové skladovací místo v obchodě.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko nový u případu 3.2.2.13.
2. Systém uživateli nabídne dialog s inputem pro zadání jména nového skladovacího místa.
3. Uživatel zadá jméno a klikne na tlačítko vytvořit.
4. Systém zkontroluje uživatelem zadané jméno a vytvoří nové skladovací místo.
5. Uživatele informuje o úspěšném vytvoření nového skladovacího místa.

3.2.2.16 Smazat skladovací místo

Popis: Odebere skladovací místo z obchodu.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko smazat, pokud má v tabulce z případu 3.2.2.13 vybrané skladovací místo.
2. Systém uživateli nabídne potvrzovací dialog.
3. Uživatel potvrdí smazání.
4. Systém zkontroluje, zda skladovací místo lze smazat a smaže ho.
5. Uživatele informuje o úspěšném smazání.

3.2.2.17 Zobrazit seznam registrovaných uživatelů

Popis: Zobrazí registrované uživatele v tabulce podle filtru.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko uživatelé v navigaci nebo na tlačítko hledat u filtru.
2. Systém načte uživatele podle fitlru do tabulky.
3. Uživateli je zobrazena tabulka s uživateli.

3.2.2.18 Upravit uživateli roli

Popis: Změní danému uživateli roli a tím přístupová práva.
Aktéři: Administrátor
Scénář:

1. Uživatel klikne na tlačítko změnit roli, pokud má v tabulce z případu 3.2.2.17 vybraného uživatele.
2. Systém uživateli nabídne dialog s roletkou s uživatelskými rolemi.
3. Uživatel vybere roli a klikne na tlačítko potvrdit.
4. Systém zkontroluje uživatelem vybranou roli a uloží upraveného uživatele.
5. Uživatele informuje o úspěšné změně role.

3.3 Návrh tříd

Základem elektronického obchodu by mělo být zboží rozdělené do kategorií. Bude potřeba, aby zboží někdo kupoval, expedoval a případně se staral o aktuální nabídku, proto zavedu uživatele. Dále ještě potřeba zavést nákupní košík a z něj vytvořené objednávky. Z těchto požadavků jsem navrhnul tyto třídy:

Commodity: Je základem obchodu, představuje nabízené zboží, kromě základních povinných parametrů (jméno, cena, počet na skladu a popisek) bude obsahovat obrázky z třídy Picture, místo uskladnění z třídy Storage, a kategorie z třídy Category, ve kterých se zboží bude nacházet.

Category: Důležitá třída sloužící k rozdělení zboží. Kategorie se budou formovat do stromu, proto kromě povinného jména bude mít i volitelného rodiče stejné třídy.

Picture: Jednoduchá třída představující obrázky, je vztažená ke zboží. Povinně obsahuje pouze bitový obsah obrázku jako blob, ale ještě má dvě vlastnosti, které upřesňují jeho chování - veřejný obrázek a výchozí obrázek.

Storage: Třída sloužící jen k představě skladovacího místa, kde se daný produkt nachází, například určitou poličku, regál. Obsahuje pouze jméno.

ShopUser: Třída představující uživatele. Povinné parametry budou jen přihlašovací jméno a heslo, ale určitě by se využilo i skutečné jméno a příjmení nebo e-mail, případně telefon. Každý uživatel bude mít přidělené určité systémové role z enumu UserRole(tři role: uživatel, zaměstnanec a administrátor). Také by měl vlastnit adresu Address.

Address: Třída představující jednoduchou adresu. Obsahuje atributy, ze kterých lze nadepsat obálku dopisu nebo balík: jméno, ulice s číslem popisným, PSČ, město a zemi.

ShoppingCart: Tato třída v sobě uchovává pro dané sezení instance třídy CartItem a představuje nákupní koš. Proto má pouze povinný atribut session.

CartItem: Třída představující zboží v košíku nebo v objednávce, obsahuje druh zboží - Commodity a jejich počet.

ShopOrder: Třída, která představuje objednávku. Parametry má dva, enum OrderStatus (představuje stav objednávky, jsou 4 stavy: nová, přijatá, odeslaná a vyřízená) a zda je zaplaceno. Dále obsahuje zboží z košíku CartItem, doručovací metodu ShippingMethod a uživatele, který objednávku zadal, ShopUser.

ShippingMethod: Třída představující doručovací metodu, například PPL, na dobírku Českou Poštou. Povinné parametry jsou název a cena, která bude připočítána k celkové úhradě.

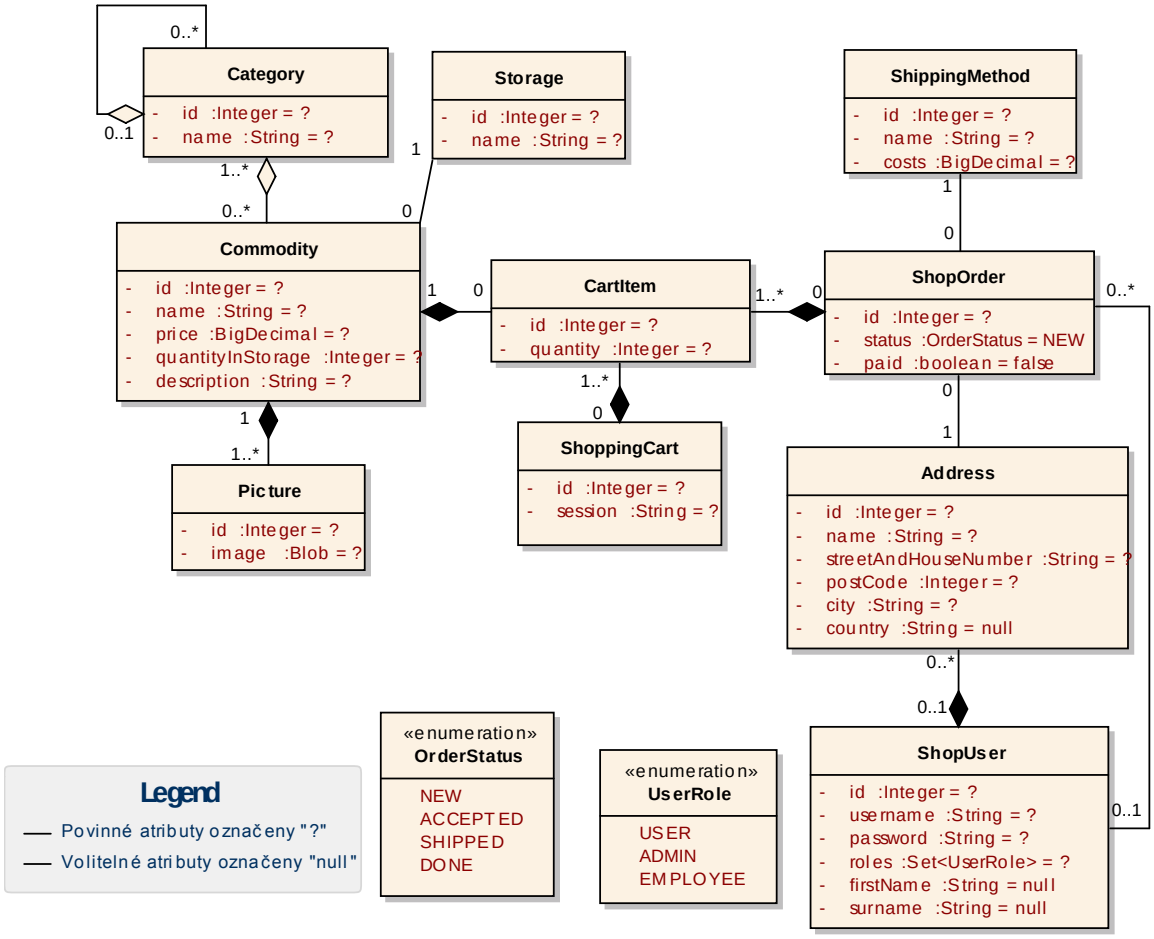
Přehledně jsou všechny tyto třídy shrnuty v UML Class Diagramu na Obrázku 3.3.

3.4 Návrh uživatelského rozhraní, popis funkcí

Protože obchod bude mít dvě hlavní části, navrhnu Graphical user interface (GUI)[4] pro každou zvlášť. Jednotlivé obrázky se kvůli přehlednosti nacházejí v příloze.

3.4.1 Návrh administrační části

Administrační část elektronického obchodu bych ještě rozdělil do dvou podsekcí.



Obrázek 3.3: UML Class Diagram

3.4.1.1 Správa objednávek

Do správy objednávek by měli přístup řadoví zaměstnanci. Tato část by měla zajišťovat snadnou práci s objednávkami. Měla by obsahovat záložku vyřízené, odeslané, nové a přijaté. Ve vyřízených jsou zahrnuty všechny objednávky, které došly k zákazníkovi v pořádku a firma za ně dostala zaplaceno, něco jako archiv objednávek. V odeslaných by byl seznam všech zakázek již na cestě k zákazníkovi, ale ještě nezaplacených. V sekci nové by byl seznam všech potvrzených objednávek od zákazníka, ale ještě nepřijatých zaměstnancem. Nakonec do části čekající bych zahrnul všechny rozpracované zakázky. Po přijetí zaměstnancem by se zakázka přesunula z nových do přijatých. Pokud byla zákazníkem při objednávání rovnou zaplacena, přidá se status zaplaceno. Jakmile zaměstnanec zkompletuje a zabalí obsah objednávky, zkontroluje způsob dopravy a platby (dobírka, PPL, kartou aj.) a připraví ji k odeslání (zde odhaduji, že jednou denně se všechny připravené zásilky předají poště nebo jiným dopravcům), tak se objednávka přesune do sekce odeslané. Ve chvíli kdy zákazník dostane

zásilku a objednávka má status zaplaceno, tak se přesune do sekce vyřízené. Návrh stránky je vidět na Obrázku A.1.

3.4.1.2 Správa obsahu obchodu

Do správy obsahu by měl přístup zaměstnanec s vyššími právy, administrátor obchodu. Obsahovala by záložku se seznamem produktů, kde bude mít možnost vytvářet, upravovat a mazat nabízené produkty. Dále záložku s kategoriemi, do kterých budou produkty v obchodu rozdělené, správu skladovacích míst a způsobů dopravy. Ve správě obsahu by také měla být stránka se seznamem uživatelů s možností měnit jejich přístupová práva. Návrh přehledu je na Obrázku A.2 a editace produktu na Obrázku A.3.

3.4.2 Návrh veřejné části

Veřejná část by se skládala především z pěti hlavních stránek: úvodní stránka, stránka kategorie, stránka s výsledky hledání, stránka s detailem produktu a košíková stránka. Veřejná část obchodu by byla přístupná nepřihlášeným uživatelům. Uživatel by se mohl kdykoliv přihlásit. Pokud by neměl účet, zaregistroval by si nový. Všechny stránky by obsahovaly navigaci skrze kategorie v podobě rozklikávacího menu a malé vyhledávací okénko. Úvodní stránka by obsahovala náhled na několik produktů ve slevě, případně nějaký text, uvítaní, cokoliv důležitého, co by měl zákazník vědět, viz Obrázek A.4.

Stránka kategorie by zobrazovala seznam produktů dané kategorie, nejlépe malý obrázek, název, cenu, kousek popisku, počet kolik zbývá na skladu a po klinutí na něj by se přešlo na jeho detailovou stránku. Také by obsahovala možnost filtru, viz Obrázek A.5.

Stránka s výsledky hledání by obsahovala seznam produktů, které by odpovídaly filtru, jehož parametry si zákazník zvolil, po kliknutí na produkt by se přešlo k jeho detailu, stejné jako u kategorie.

Stránka s detailem produktu by zobrazila daný produkt s jeho obrázky, popiskem, cenou, množstvím na skladu, jeho hodnocením oblíbenosti a tlačítkem přidat do košíku. Hodnotit produkt by mohl pouze přihlášený uživatel, ale produkt do košíku by si mohl přidat kdokoliv. Také by obsahovala náhled na 3 doporučené produkty, které by se vybraly adaptací sortimentu a zákazník by rovnou mohl přejít na jejich detail. Návrh na Obrázku A.6.

Obsah košíku by byl zobrazen na košíkové stránce, bylo by možné z něj již přidané věci odebrat, případně měnit počty produktů. Zobrazovala by se celková cena košíku. Při stisku tlačítka objednat by byl zákazník, pokud nebyl přihlášen, vyzván k přihlášení, případně vyplnění doručovacího formuláře. Tam by si také vybral způsob platby, způsob doručení, případně další specifikace. Mockup košíku je na Obrázku A.7.

3.5 Návrh adaptace

V této sekci navrhnu část obchodu týkající se adaptace. Rozhodl jsem se sledovat jak zboží, tak uživatele, proto i tato sekce bude rozdělena na dvě části.

3.5.1 Adaptace podle uživatele

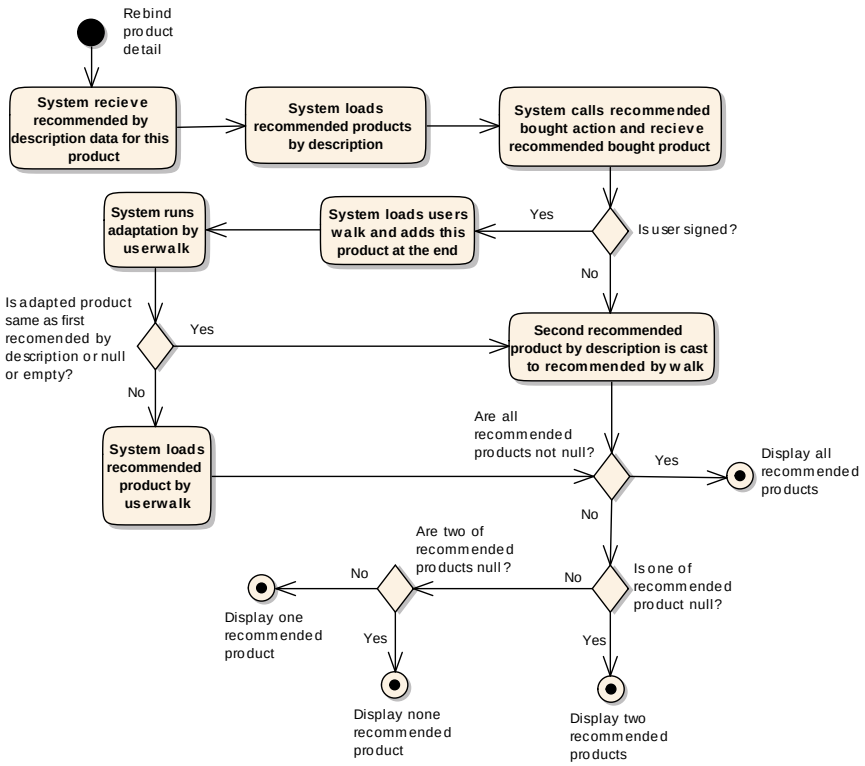
Sledováním chování uživatele budu schopen odhadnout jeho přání a úpravou nabízeného obsahu mu usnadnit průchod obchodem. Rozhodl jsem se konkrétně pro sledování pořadí kroků při daném sezení přihlášeného uživatele. Bude se sledovat pořadí navštívených produktů. Pokud jsme daný produkt již dříve navštívili, je možné nabídnout produkt, který bude pravděpodobně následovat a ušetřit tím uživatele od zbytečného klikání při hledání daného produktu. Druhým aspektem, pro který jsem se rozhodl, je sledování oblíbenosti daného produktu u zákazníků zavedením hodnotícího systému. Zde je vyžadována aktivní účast registrovaného uživatele, který bude moci ohodnotit dané zboží podle své spokojenosti. Oblíbenost využiji jako druhý filtr pro podobné produkty. Další věc, která by šla sledovat, bude, jaké zboží si objednali ostatní také s tímto produktem. Zde v objednávkách obsahujících daný produkt najdu další nejčetnější produkt, který si zákazníci také koupili a doporučím ho.

3.5.2 Adaptace podle zboží

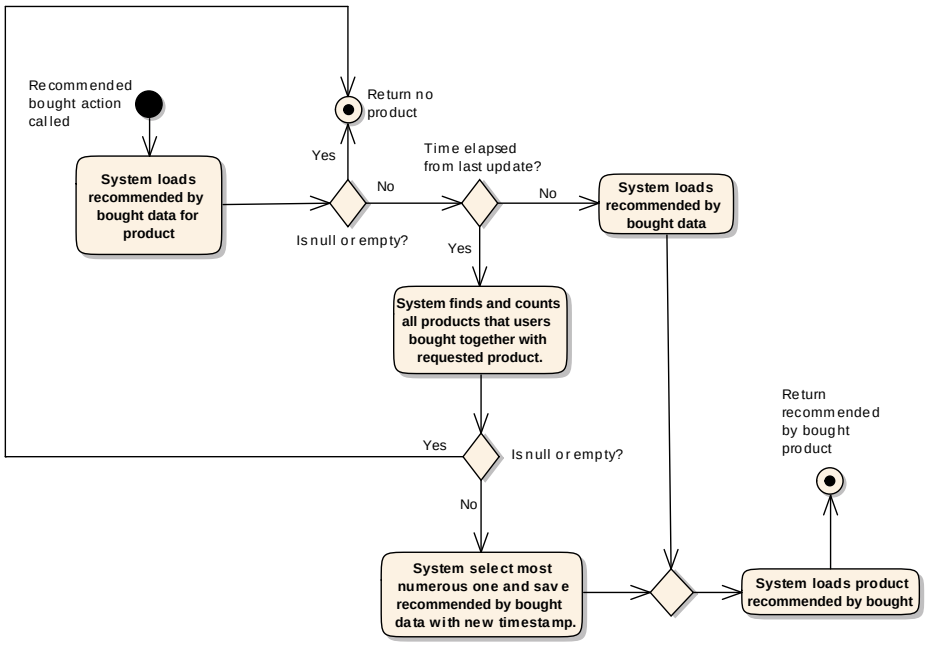
Sledováním určitých statistik a vlastností budu schopen zákazníkovi upravit zobrazený obsah tak, aby nabízel a doporučoval nejkvalitnější a nejpodobnější zboží. Konkrétně jsem se rozhodl pro podobnost popisků. V obchodu najdu 3 produkty s nejpodobnějším popiskem a vyberu z nich ten nejlépe hodnocený. Tento produkt pak mohu nabídnout jako doporučený.

3.5.3 Adaptační aktivity diagramy

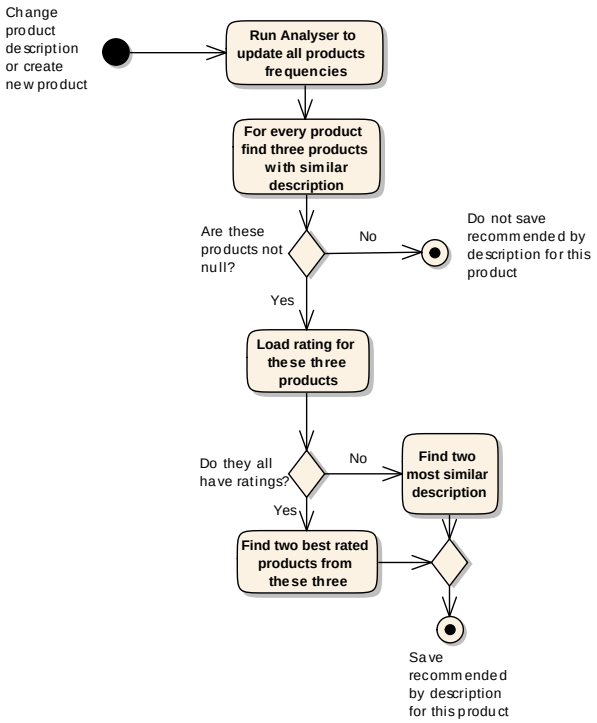
Jak bude probíhat adaptování na stránce produktu je vidět na Obrázku 3.4. Práce s doporučením, co ostatní zákazníci koupili také, je na Obrázku 3.5. Příprava dat pro adaptaci podle podobného popisku je na Obrázku 3.6.



Obrázek 3.4: Activity diagram - adaptace na stránce produktu



Obrázek 3.5: Activity diagram - adaptace 'ostatní koupili také'



Obrázek 3.6: Activity diagram - adaptace podle popisku

Kapitola 4

Implementace obchodu

4.1 Nastavení projektu pro Google App Engine

GAE po správném nastavení je schopen podporovat i různé frameworky, já budu využívat Spring[11], PrimeFaces[5] a ASF[7]. K přístupu k datům budu využívat JPA[8]. Jako vývojové prostředí používám NetBeans 7.3.1. Pro ty neexistuje oficiální GAE plugin. Poslední NetBeans plugin pro GAE zmíněný na oficiálních stránkách je pro verzi 7.2[1]. Pro verzi 7.3.1[2] lze použít upravený plugin. Bohužel tyto pluginy fungují bez problému jen pro Appengine Web Projecty, pokud chci používat Apache Maven[1] jsem odkázán na App Engine Maven Plugin. Pravděpodobně lepší podporu má vývojové prostředí Eclipse, o jehož plugin se stará sám Google.

4.1.1 App Engine Maven Plugin

Díky tomu, že používám Maven, nemohu používat NetBeans pluginy. Proto jako první zajistím nastavení App Engine Maven Pluginu. Ten pro fungování vyžaduje nyní verzi Mavenu 3.1 nebo vyšší. Já používám momentálně Maven verze 3.1.1 a nenarazil jsem zatím na žádný problém. GAE Maven plugin mívá stejnou verzi jako SDK. Aktuální je 1.8.8, ale celkem často se mění. Dříve důležitá věc, která zásadně ovlivňovala chování testů, bylo zapnutí High Replication Datastore (HRD)[2] na lokálním datastore. Jedna z potřebných věcí ke zprovoznění transakcí na více než jedné skupině entit. Nyní je již zapnuto výchozím nastavením, kde 10% zápisů není ihned viditelných v globálních queries. Já to pro jistotu nechávám nastaveno na 20%, jak jsem to měl dříve, když to ještě nemělo výchozí nastavení. Procenta, případně další vlastnosti lokálního datastore, se nastavují pomocí jvmFlag. Nyní plugin můžeme přidat do části plugins v pom.xml:

```
<plugin>
    <groupId>com.google.appengine</groupId>
    <artifactId>appengine-maven-plugin</artifactId>
    <version>${gae-maven-plugin.version}</version>
    <configuration>
      <jvmFlags>
```

[1] https://kenai.com/projects/nbappengine/pages/Home
[2] https://code.google.com/p/nb-gaelyk-plugin

```
            <jvmFlag>-Ddatastore.default_high_rep_job_policy_unapplied_job_pct=20
            </jvmFlag>
        </jvmFlags>
    </configuration>
</plugin>
```

Ovládání GAE pluginu je popsáno v příloze C.2.

4.1.2 Přidání povinných GAE balíčků

Plugin by sám o sobě nebyl moc platný, pokud by projekt neobsahoval důležité GAE balíčky. Mezi vyžadované balíčky patří: appengine-api-1.0-sdk a servlet-api. Druhý zmíněný musí mít scope provided. Také je povinné nastavení Maven packaging typu na hodnotu war(<packaging>war</packaging>). Další balíčky jsou volitelné. Určitě využiji testování projektu. Testování vyžaduje minimálně balíčky junit, mockito-all, appengine-testing a appengine-api-stubs. Také je důležité, aby byl scope nastaven na hodnotu pro testování - test. Existují i další balíčky ale ty momentálně nevyužiji. Například pokud bych chtěl používat experimentální API, tak přidám appengine-api-labs.

```
<dependency>
        <groupId>com.google.appengine</groupId>
        <artifactId>appengine-api-1.0-sdk</artifactId>
    <version>${gae.version}</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
<dependency>
        <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>1.9.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.google.appengine</groupId>
    <artifactId>appengine-testing</artifactId>
    <version>${gae.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
        <groupId>com.google.appengine</groupId>
        <artifactId>appengine-api-stubs</artifactId>
```

```
    <version>${gae.version}</version>
    <scope>test</scope>
</dependency>
```

4.1.3 Zprovoznění Java Persistance API

Pro práci s daty pomocí JPA 2 na GAE je potřeba DataNucleus. Abych mohl objekty ukládat a načítat z úložiště, musí být projekt po kompilaci obohacen DataNucleus enhancerem. To zajistím přidáním Maven DataNucleus pluginu. Pro JPA 2 je nutné používat AppEngine DataNucleus verze 2.x, který pracuje s DataNucleus Access Platform verze 3.x (Pro JPA 1 Datanucleus verze 1.x, je nutné dodržet tuto kombinaci verzí, protože nejsou plně zpětně kompatibilní[3]). Já použiji verzi 2.1.2. Samotný plugin nastavím, aby obohacoval (enhance) třídy podle Datanucleus návodu[4]:

```
<plugin>
        <groupId>org.datanucleus</groupId>
        <artifactId>datanucleus-maven-plugin</artifactId>
        <version>3.2.0-m2</version>
        <configuration>
                <api>JPA</api>
                <persistenceUnitName>transactions-optional</persistenceUnitName>
                <verbose>true</verbose>
                <targetDirectory>target/classes</targetDirectory>
        </configuration>
        <executions>
                <execution>
                <phase>process-classes</phase>
            <goals>
                <goal>enhance</goal>
            </goals>
        </execution>
    </executions>
        <dependencies>
            <dependency>
                <groupId>org.datanucleus</groupId>
                <artifactId>datanucleus-api-jpa</artifactId>
            <version>${datanucleus-api-jpa.version}</version>
        </dependency>
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>datanucleus-core</artifactId>
            <version>${datanucleus-core.version}</version>
        </dependency>
    </dependencies>
</plugin>
```

Kromě nastavení pluginu je třeba přidat DataNucleus balíčky pro runtime. Konkrétně datanuclues-core a datanuclues-api-jpa, oboje mi funguje s verzí 3.1.2. Samozřejmě je také potřeba přidat samotné JPA. Balíček geronimo-jpa_2.0_spec verze 1.1 postačí.

```
<dependency>
        <groupId>com.google.appengine.orm</groupId>
    <artifactId>datanucleus-appengine</artifactId>
    <version>2.1.2</version>
</dependency>
<dependency>
    <groupId>org.apache.geronimo.specs</groupId>
    <artifactId>geronimo-jpa_2.0_spec</artifactId>
    <version>1.1</version>
</dependency>
<dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>datanucleus-core</artifactId>
    <version>${datanucleus-core.version}</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>datanucleus-api-jpa</artifactId>
```

[3] https://developers.google.com/appengine/docs/java/datastore/jpa/overview-dn2
[4] http://www.datanucleus.org/products/accessplatform/jpa/maven.html

```
    <version>${datanucleus-api-jpa.version}</version>
    <scope>runtime</scope>
</dependency>
```

Aplikace s JPA nebude fungovat, pokud není nastaven soubor persistence.xml v META-INF. Použiji úplně základní nastavení z dokumentace GAE[2], to k funkci JPA2 na GAE stačí, ale pro správnou funkci Datanuclues enhancer je dobré specifikovat, jaké třídy se budou vztahovat k Datastoru a ostatní vyřadit pomocí exclude-unlisted-classes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
    <persistence-unit name="transactions-optional">
        <provider>org.datanucleus.api.jpa.PersistenceProviderImpl</provider>
        <class>cz.cvut.fel.asf.adapt.gomawe.storage.gae.GaeUserModelAttribute</class>
        <class>cz.cvut.fel.asf.adapt.gomawe.storage.gae.GaeContentUnitModelAttribute</class>
        <class>cz.cvut.fel.asf.adapt.gomawe.storage.gae.GaeUserProfileAttribute</class>
        <class>cz.cvut.fel.asf.adapt.gomawe.storage.gae.AbstractPersistable</class>
        <class>cz.cvut.fel.adaptiveshop.adaptation.ShopUserWalk</class>
        <class>cz.cvut.fel.adaptiveshop.model.Commodity</class>
        <class>cz.cvut.fel.adaptiveshop.model.Category</class>
        <class>cz.cvut.fel.adaptiveshop.model.Picture</class>
        <class>cz.cvut.fel.adaptiveshop.model.ShopUser</class>
        <class>cz.cvut.fel.adaptiveshop.model.Storage</class>
        <class>cz.cvut.fel.adaptiveshop.model.SuperCommodity</class>
        <class>cz.cvut.fel.adaptiveshop.model.SuperCategory</class>
        <properties>
            <property name="datanucleus.NontransactionalRead" value="true"/>
            <property name="datanucleus.NontransactionalWrite" value="true"/>
            <property name="datanucleus.ConnectionURL" value="appengine"/>
            <property name="datanucleus.singletonEMFForName" value="true"/>
            <property name="datanucleus.appengine.datastoreEnableXGTransactions" value="true"/>
        </properties>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
    </persistence-unit>
</persistence>
```

4.1.4 Zprovoznění Java Server Faces

4.1.4.1 JSF 2.1

Tato část byla složitější. JSF jsou potřeba pro fungování frameworku PrimeFaces. Samotné přidání artefaktů pro funkci JSF 2.1 nestačilo, proto jsem postupoval podle návodu pro Eclipse[3] a upravoval kroky.

1. Balíček jboss-el lze získat podle návodu ze Seam 2 frameworku. Na CD se nachází ve složce pre-install.
2. Je třeba ho nainstalovat do lokálního Maven repositáře. V projektu je instalace součástí install-dependency.bat ve složce pre-install.
3. Přidat balíčky jboss-el, javax.el-api a javax.faces do pom.xml.
4. Změnit Unified Expression Language ve web.xml na jboss.
5. Kvůli GAE vypnout threading ve web.xml.
6. Kvůli ViewExpiredException změnit STATE_SAVING_METHOD na clienta.
7. Nakonec je třeba přidat faces-config.xml soubor do složky WEB-INF.

Ukázka kroků týkajících web.xml:

```
<context-param>
    <param-name>com.sun.faces.expressionFactory</param-name>
    <param-value>org.jboss.el.ExpressionFactoryImpl</param-value>
</context-param>
<context-param>
        <param-name>com.sun.faces.enableThreading</param-name>
    <param-value>false</param-value>
</context-param>
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
</context-param>
```

Když jsem JSF nastavoval poprvé, bylo ještě třeba přidat třídu WebConfiguration z balíčku JSF a zakomentovat obsah metody processJndiEntries a importy javax.naming.Context, javax.naming.InitialContext a javax.naming.NamingException. Tyto třídy se nenacházejí na whitelist[5] povolených tříd pro GAE. Při kompilaci se třída z balíčku přepsala upravenou třídou a GAE fungovalo správně. Nakonec díky FileUpload a PrimeFaces 4 jsem nucen používat verzi JSF 2.1.26 místo 2.2.4, protože GAE nepodporuje servlet 3.0, který tento FileUpload používá. Pokud ale není třeba nahrávat soubory, tak JSF 2.2.4 fungovalo. Pro verzi 2.1.26 odpadla nutnost upravovat třídu WebConfiguration. Artefakty pro pom.xml budou tedy vypadat následovně:

```
<dependency>
        <groupId>org.glassfish</groupId>
        <artifactId>javax.faces</artifactId>
        <version>2.1.26</version>
</dependency>
<dependency>
        <groupId>javax.el</groupId>
        <artifactId>el-api</artifactId>
        <version>2.2.5</version>
</dependency>
<dependency>
        <groupId>org.jboss.el</groupId>
        <artifactId>jboss-el</artifactId>
        <version>2.2</version>
</dependency>
```

4.1.4.2 JSF 2.2

Při pokusu o update na JSF 2.2.4, jsem se dozvěděl, proč se tak složitě přidává balíček jboss-el. Problém spočívá v tom, že GAE změnilo pořadí načítání tříd, aby je přepsalo jako poslední. GAE má vlastní starý el balíček od com.sun. Proto pokud bych chtěl přidat nový el balíček od com.sun, tak třídy, které obsahuje, ve finále stejně budou přepsány těma z balíčku GAE. GAE balíček jboss-el neobsahuje, takže pokud změním výchozí expressionFactory na tu od jboss v konfiguraci web.xml, tak ho GAE nepřepíše a já mám k dispozici stejné třídy, jaké jsem dodal. JSF 2.2.4 fungovalo v konfiguraci s balíčkem jboss-el verze 2.2 místo doporučovaného javax.el 3.0.0, javax.faces verze 2.2.4 a javax.el-api verze 3.0.0.

4.1.5 Další nastavení souborů

Soubor, bez kterého se ani nepovede aplikaci na GAE nahrát, je appengine-web.xml, který se nachází ve WEB-INF složce. Obsahuje unikátní identifikátor aplikace, který je také součástí domény, pro můj projekt jsem zvolil czcvutfeladaptiveshop. Také obsahuje verzi aplikace a pro základní aplikace se zapnou pouze sessions a threadsafe(povolí GAE posílání více dotazů na server najednou). Nepovažuji za nutné mít zapnuté asynchronní ukládaní sessions do datastore a memcache:

[5](https://developers.google.com/appengine/docs/java/jrewhitelist

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
    <application>czcvutfeladaptiveshop</application>
    <version>1</version>
    <threadsafe>true</threadsafe>
    <sessions-enabled>true</sessions-enabled>
    <async-session-persistence enabled="false" />
</appengine-web-app>
```

Ještě jsem v průběhu implementace potřeboval nastavit datastore indexy a použít task queues [2]. Pro datastore indexy se do WEB-INF přidá soubor datastore-indexes.xml, který obsahuje vyžadované indexy. Pokud app engine vyžaduje nějaký index, tak při vyhození informační výjimky i navrhne jeho podobu. Takže pokud nějaký chybí, tak ho stačí pouze překopírovat do konfiguračního souboru. Nastavení Task Queues je také celkem jednoduché. Konfigurační soubor queue.xml přidáme také do WEB-INF složky. Jako příklad uvedu soubory, které používám v aplikaci queue.xml. Rate určuje, jak často se kontroluje fronta. Mám to nastaveno na každou sekundu. Retry-limit určuje, kolikrát se maximálně spustí daná úloha, pokud je nějaká chyba. V příkladu to zkusím maximálně jednou:

```
<?xml version="1.0" encoding="UTF-8"?>
<queue-entries>
  <queue>
    <name>queue</name>
    <rate>1/s</rate>
    <retry-parameters>
      <task-retry-limit>1</task-retry-limit>
    </retry-parameters>
  </queue>
</queue-entries>
```

A datastore-index.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<datastore-indexes
  autoGenerate="true">
    <datastore-index kind="Commodity" ancestor="false" source="manual">
        <property name="quantity" direction="asc" />
        <property name="name" direction="asc" />
    </datastore-index>
    <datastore-index kind="ShopUserWalk" ancestor="true" source="manual">
        <property name="dateStamp" direction="desc" />
    </datastore-index>
    .
    .
</datastore-indexes>
```

4.1.6 Zprovoznění AspectJ

Toto sice není úplně čistě GAE problém, ale GAE tento proces ovlivní. GAE nepodporuje load-time-weaving, takže jsem nucen používat compile-time-weaving. Od toho se odvíjí nastavení applicationContext.xml a pom.xml. Vyžaduje se přidání balíčků a pluginů s aspecty. Konkrétně to jsou spring-aspects, aspectjrt a pro zabezpečení spring-security-aspects. Spring-aspects a spring-security-aspects mají scope compile.A pluginy jsou aspectj-maven-plugin[6] a maven-compiler-plugin[7]. Maven-aspectj-plugin potřebuje nastavit dva cíle, pokud chceme využívat i testy. Také vyžaduje dodání balíčků s aspecty přímo a závislost aspectjweaver. Výsledné nastavení pluginů:

[6] http://mojo.codehaus.org/aspectj-maven-plugin/compile-mojo.html
[7] http://maven.apache.org/plugins/maven-compiler-plugin/

```
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>aspectj-maven-plugin</artifactId>
<version>1.5</version>
<executions>
    <execution>
        <id>compile</id>
        <configuration>
            <source>1.7</source>
            <target>1.7</target>
            <complianceLevel>1.7</complianceLevel>
            <aspectLibraries>
                <aspectLibrary>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-aspects</artifactId>
                </aspectLibrary>
                <aspectLibrary>
                    <groupId>org.springframework.security</groupId>
                    <artifactId>spring-security-aspects</artifactId>
                </aspectLibrary>
            </aspectLibraries>
        </configuration>
        <goals>
            <goal>compile</goal>
        </goals>
    </execution>
    <execution>
        <id>test-compile</id>
        <configuration>
            <source>1.7</source>
            <target>1.7</target>
            <complianceLevel>1.7</complianceLevel>
            <aspectLibraries>
                <aspectLibrary>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-aspects</artifactId>
                </aspectLibrary>
                <aspectLibrary>
                    <groupId>org.springframework.security</groupId>
                    <artifactId>spring-security-aspects</artifactId>
                </aspectLibrary>
            </aspectLibraries>
        </configuration>
        <goals>
            <goal>test-compile</goal>
        </goals>
    </execution>
</executions>
<dependencies>
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>${aspectj.version}</version>
    </dependency>
</dependencies>
</plugin>
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
        <source>1.7</source>
        <target>1.7</target>
            <compilerArguments>
                <endorseddirs>${endorsed.dir}</endorseddirs>
        </compilerArguments>
        <showDeprecation>true</showDeprecation>
    </configuration>
</plugin>
```

4.2 Použité frameworky

4.2.1 Spring Framework

Jeden z požadavků je použití frameworku Spring[11]. Pomocí něj se mohu postarat o přístup k datům s využitím JPA[8], spojení závislostí pomocí anotací @Autowire, zavedení MVC modelu, zabezpečení pomocí security a mnoho dalšího. Protože byla potřeba Dependecy Injection i s objekty mimo Spring kontejner tvořenými operátorem new, musel jsem zprovoznit

anotaci @Configurable[8] . K tomu je nutné mít funkční AspectJ(viz 4.1.6) time weaving . Spring je nastaven v application-context.xml kromě klasického nastavení a zapnutí anotací je potřeba kvůli AspectJ zapnout proxy-target-class, u transaction-manager nastavit mód na aspectj a zapnout @PeristanceUnit anotace:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
    <aop:aspectj-autoproxy proxy-target-class="true"/>
    <tx:annotation-driven transaction-manager="transactionManager" mode="aspectj" />
    <context:component-scan base-package="cz.cvut.fel.adaptiveshop"/>
    <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value=""/>
        <property name="suffix" value=".xhtml"/>
    </bean>
    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean"
    lazy-init="true" >
        <property name="persistenceUnitName" value="transactions-optional"/>
    </bean>
    <bean name="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory"/>
    </bean>
    <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>
</beans>
```

4.2.2 PrimeFaces a JSF

Tyto frameworky mi zajišťují správu uživatelského rozhraní. Používám úplně nové PrimeFaces 4.0 [5] a JSF 2.1.26. Je potřeba nastavit faces-config.xml, ve kterém nastavím můj vlastní view-hanler, který jsem byl nucen napsat kvůli ajax eventům. Pokud jsou v adrese některé parametry jako GET, formuláře mají stále action adresu původní a ajax eventy nefungují. Také je třeba přidat el-resolver pro Spring:

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
              xmlns="http://xmlns.jcp.org/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
              http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <application>
        <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>
    </application>
    <application>
        <view-handler>cz.cvut.fel.adaptiveshop.web.MyViewHandler</view-handler>
    </application>
</faces-config>
```

4.2.3 Adaptive System Framework

ASF [7] byl vytvořen, aby pomohl softwarovým vývojářům s tvorbou adaptivních webových aplikací. Já využiji knihovny ASF.Core, ASF.Persistence-GAE a ASF.Web-Primefaces (popis knihoven 4.4) verze 0.3-SNAPSHOT. Popíši zde nastavení, které jsem musel přidat do applicationContext, abych vytvořil funkční AdaptationManager.

[8] http://docs.spring.io/spring/docs/3.0.0.RC2/reference/html/ch07s08.html

```
<bean id="gaeUserProfileAttributeDAO"
      class="cz.cvut.fel.asf.adapt.gomawe.storage.gae.GaeUserProfileAttributeDAOImpl" />
<bean id="gaeUserModelAttributeDAO"
      class="cz.cvut.fel.asf.adapt.gomawe.storage.gae.GaeUserModelAttributeDAOImpl" />
<bean id="adaptationManager"
      class="cz.cvut.fel.asf.adapt.gomawe.AdaptationManager" >
        <property name="userProfileAttributeDAO" ref="gaeUserProfileAttributeDAO" />
        <property name="userModelAttributeDAO" ref="gaeUserModelAttributeDAO" />
        <property name="contentUnitModelAttributeDAO" ref="shopContentUnitModelAttributeDAOImpl" />
        <property name="userDAO" ref="shopUserDAOImpl" />
        <property name="userProfileImplementationClass"
          value="cz.cvut.fel.adaptiveshop.adaptation.ShopUserProfile" />
        <property name="userModelImplementationClass"
          value="cz.cvut.fel.adaptiveshop.adaptation.ShopUserModel" />
        <property name="contentUnitModelImplementationClass"
          value="cz.cvut.fel.adaptiveshop.adaptation.ShopContentUnitModel" />
</bean>
<bean id="datastorageDAO"
      class="cz.cvut.fel.asf.persistence.gae.StorageDAO" />
```

A tyto závislosti jsou potřebné v pom.xml:

```
<dependency>
        <groupId>cz.cvut.fel.asf</groupId>
    <artifactId>asf-core</artifactId>
    <version>${asf-version}</version>
</dependency>
<dependency>
        <groupId>cz.cvut.fel.asf</groupId>
    <artifactId>asf-persistance-GAE</artifactId>
    <version>${asf-version}</version>
</dependency>
    <dependency>
    <groupId>cz.cvut.fel.asf</groupId>
    <artifactId>asf-web-primefaces</artifactId>
    <version>${asf-version}</version>
</dependency>
```

4.3 Implementace pomocí MVC

4.3.1 Model

Při implementaci jsem nakonec udělal několik odchylek od návrhu. Všechny třídy, které se ukládají do datastore musí rozšiřovat abstraktní třídu AbstractPersistable z knihovny ASF.Persistence.GAE (viz 4.4.2) a obsahovat anotaci @Entity.

SuperCommodity vznikla jen díky GAE problému s transakcemi na více skupinách entit, limit GAE je pět skupin a každá entita uložená bez předka se považuje za novou skupinu. Řešení spočívá v tom, že vytvořím jedinou entitu třídy SuperCommodity a použiji ji jako rodiče při vytváření nových entit třídy Commodity. Tím zajistím, že všechny instance Commodity budou potomci stejné entity a tudíž i budou ve stejné skupině entit. Nyní klidně budu moci používat transakce na více než pěti entitách. Proto má pouze vztah OneToMany s Commoditou a jméno, které použiji v konstruktoru.

Stejný princip použiji i u třídy Category, takže vznikne SuperCategory. U ostatních modelových tříd jsem díky fetchType LAZY zatím nenarazil na problém, že by se překročil počet pět skupin entit v jedné transakci. Proto jsou zatím bez super rodiče, pokud by v budoucnu došlo k rozšíření aplikace, které by je vyžadovalo, nebude problém je dopsat.

Třída Commodity implementuje rozhraní IRateable z knihovny ASF.Core, které poté umožňuje adaptaci pomocí třídy SimilarByContentLinkAdaptationAlgorithm, jejíž funkce je blíže popsána v adaptační části 4.4. Commodity je spojena vztahem OneToMany jako rodič s třídou reprezentující obrázky Picture. K tomu obsahuje ještě jeden základní obrázek, který používám k vykreslování náhledů. Podobně je to s třídou reprezentující kategorie Category.

Zde je také vztah OneToMany, ale už musí mít anotaci @Unowned, protože kategorie jsou již uložené v datastore se svým rodičem SuperCategory, navíc by bylo nevhodné, aby třída Commodity byla rodič třídy Category. Také vztah k třídě Storage musí být jednosměrný @Unowned ManyToOne.

Každá entita z Picture spadá do stejné skupiny entit jako Commodity, protože je to její potomek. Přidal jsem ještě nepovinnou vlastnost isPublic, která určuje, zda se obrázek zobrazuje zákazníkům v galerii , a isDefault, která určuje, zda právě tento obrázek je u Commodity nastaven jako výchozí a zobrazován v náhledu. Každá Commodity musí mít v setu obrázků právě jeden, který je výchozí a ten musí být samozřejmě veřejný.

Třída Category reprezentuje kategorie uspořádané do stromu. Toho docílím tak, že kromě společného rodiče SuperCommodity, vytvořím ještě jeden @Unowned vztah ManyToOne ke Category, který bude představovat rodiče ve stromu. Tento nadbytečný vztah přidám kvůli možnosti editace struktury stromu. Pokud chci vytvořit pevný strom, bez možnosti editace, tak nepotřebuji SuperCategory, ale prostě vytvořím kořenovou entitu bez rodiče a od ní budu dále větvit další entity. Ale v GAE jakmile uložím jednou entitu s rodičem, tak ho již nelze změnit, protože rodič je zakódován v primárním klíči. To by asi pro obchod nebylo dobré, kdybych neměl možnost si s kategoriemi hýbat dle mého uvážení. Proto jsem zvolil druhého rodiče, který určuje podobu stromu a mohu ho libovolně měnit.

Třída ShopUser je stejná jako v návrhu, pouze implementuje rozhraní IUser z knihovny ASF.Core, které je vyžadováno adaptačními třídami a neobsahuje adresu.

Enum uživatelských rolí UserRole implementuje rozhraní GrantedAuthority z balíčku org.springframework.security.core kvůli lehčí práci se Spring Security. Navíc jsem místo EMPLOYEE přidal možnost blokace BLOCKED.

Třída Storage je stejná jako v návrhu.

Rozhraní, která tvoří Data Access Object vrstvu rozšiřují rozhraní IDAO z knihovny ASF.Core. Obsahují definice dalších metod, které jsou většinou filtry nebo metoda na vytvoření nové entity s povinnými parametry.

Třídy, které kromě těchto DAO implementují rozhraní ICanDelete z knihovny ASF.Core, rozšiřují třídu AbstractDAO z knihovny ASF.Persistence.GAE. Všechny třídy musí obsahovat anotaci @Repository. Všechny metody, které jsem definoval v rozhraních nyní implementuji buď pomocí metod z AbstractDAO nebo pomocí Query. Také se zde nachází kontroly pomocí business check z ASF (popis 4.4.1.4).

Kromě DAO používám k získávání dat ještě rozšíření třídy QueryByJPQL z knihovny ASF.Persistance.GAE.

Podobně jako DAO je na tom servisní vrstva. Po vytvoření rozhraní s definicí potřebných metod je implementuji v třídách, které obsahují anotaci @Service. Každá služba je spojena s potřebným DAO pomocí anotace @Autowired. Implementace metod mají anotaci @Transactional zajišťující bezpečnou práci s daty uvnitř transakcí.

4.3.2 Controller

Všechny kontrolery implementují rozhraní AbstractController z knihovny ASF.Web.Primefaces a mají anotaci @Controller. S potřebnými službami je spojuje anotace @Autowire. Zpracovávají a upravují data získaná jak z datastore, tak nahraná od uživatelů.

Nejzajímavější kontroler je DetailController, který se stará o stránku s detailem produktu ve veřejné části. Na této stránce se nachází adaptace doporučených produktů. Tento kontroler proto pracuje s adaptačními třídami. Podrobný popis implementace adaptace se nachází v části 4.4 Také se stará o hodnocení uživatelů a prodávání zboží.

4.3.3 View

Slouží k interakci s uživatelem. Prezentují se zde data získaná z kontrolerů. Data z formulářů vyplněná uživateli, se předávají ke zpracování kontrolerům. Pomocí JSF/facelets jsem vytvořil dvě šablony, které používám pro administrační a veřejnou část. Hlavní použité PrimeFaces komponenty:

- dataTable - používám k zobrazení všech seznamů, často s modem umožňující výběr řádků.
- commandButton - většina tlačítek je tvořena touto komponentou, někdy se zapnutým ajaxem.
- tree - používám k vykreslení navigace ve veřejné části, se zapnutým drag and drop ke správě Category.
- menuBar - slouží k navigaci v záhlaví a k akčním tlačítkům nad tabulkami.
- menu - slouží k navigaci v administrační části.
- graphicImage - zajišťuje vykreslování obrázků.
- galleria - zajišťuje vykreslování více obrázků u produktu.
- dialog, confirmDialog - slouží k potvrzování některých akcí, případně k zadávání krátkých údajů, hodnocení.
- rating - zobrazuje hodnocení pomocí hvězdiček.
- fileUpload - slouží k nahrávání obrázků produktů.
- ajax - odesílá ajax eventy, nejčastěji označení řádky v tabulce.
- growl - slouží k zobrazení informačních zpráv pro uživatele.

Ještě využívám vstupy inputText, password, spinner, selectBooleanCheckbox, selectOneMenu, inputTextArea a výstupy outputText, outputLabel. Z komponent JSF využívám nejčastěji event, param a panelGrid.

4.4 Adaptace

4.4.1 Knihovna ASF.Core

Tato knihovna představuje jádro ASF. K adaptaci využívám balíčky spadající pod Generic Ontological Model for Adaptive Web Environments (GOMAWE)[7]. Také je zde balíček obsahující abstraktní třídy a rozhraní k práci s daty - pesistence. Poté ještě využiji balíček s business checks.

4.4.1.1 Balíček gomawe

Obsahuje třídu AdaptationManager. Tato třída funguje jako továrna modelů, a proto je navrhnuta jako singleton. V applicationContext.xml je třeba přiřadit jí DAO pro jednotlivé druhy modelů. Pokud chceme user model (popsán v 4.4.1.2) vybraného uživatele, stačí zavolat metodu getUserModel, ve které uživatele předáme manageru a pomocí generiky nastavíme třídu rozšiřující UserModel. Stejně se postupuje i u user profilu. Pro získání modelu objektu z aplikace, stačí, aby daný objekt implementoval rozhraní IPersistable. Poté ho také můžeme předat do metody getContentUnitModel a pomocí generiky nastavit třídu rozšiřující ContentUnitModel.

4.4.1.2 Balíček gomawe.storage

Tento balíček obsahuje abstraktní třídy modelů. Tyto modely jsou tři a každý má v balíčku také své rozhraní pro AttributeDAO, které slouží k definici metod potřebných pro práci s atributy daných modelů. Implementace těchto abstraktních tříd z jádra ASF se starají o správu odpozorovaných, spočítaných, uživateli nastavených adaptačních vlastností a parametrů. Každý model má na starosti jinou část.

Implementace UserModelu by se měla starat o parametry, které aplikace odpozorovala z chovaní daného uživatele a vztahují se k modelovým třídám aplikace. Uživatel o jejich shromaždovaní ani nemusí vědět (například seznam koupeného zboží daným uživatelem), ale i může (například hodnocení spokojenosti se zakoupeným produktem). Tyto získané informace se ukládají do datového úložiště jako entity implementace UserModelAttribute.

Implementace UserProfile by se měla starat o parametry, které si uživatel sám nastaví na základě jeho preferencí. Příklady mohou být, že daný uživatel má rád dané barevné schéma nebo preferuje zobrazování menších tabulek o 10 řádcích s možností stránkování před dlouhými listy 50 položek. Tyto vlastnosti by měl být uživatel schopen nastavit ve správě jeho profilu a vztahují se pouze k danému uživateli. Obdobně se nastavení ukládá do datového úložiště jako entity třídy UserProfileAttribute.

Konečně implementace ContentUnitModel by se měla starat o parametry, které se vztahují pouze k dané modelové entitě. Jako příklady uvedu ukládání celkového počtu prodaných kusů daného produktu nebo jeho průměrná oblíbenost z hodnocení uživatelů. Také se nastavení ukládá do datového úložiště jako entity třídy ContentUnitModelAttribute.

Tyto implementace záleží na datovém úložišti. V mém projektu použiji knihovnu ASF.Persistence.GAE, ale ASF také obsahuje další druhy, například JPA, Empire nebo Hibernate.

Také tento balíček obsahuje třídu LinkObject a rozhraní ILink, které tato třída implementuje. Objekty ve vyhodnocovacích metodách musí rozšiřovat ILink a k tomu se hodí třída LinkObject, která v sobě může uchovat jakýkoliv objekt.

4.4.1.3 Balíček gomawe.reasoning.linkadaptation

V tomto balíčku jsou momentálně dvě třídy, které implementují ILinkGroupAdaptationAlgorithm a tím i IAdaptationAlgorithm.

Třída SimilarByContentLinkAdaptationAlgorithm najde "K" nejbližších sousedů předaného objektu z předané kolekce objektů. Objekty musí implementovat ILink, proto využívám LinkObject. Objekt v ILink musí implementovat rozhraní IRateable a IPersistable. Proto mohu v tomto algoritmu používat Commodity obalené v LinkObejct. Objekt, ke kterému se adaptace vztahuje, se předává v konstruktoru nebo v metodě adapt spolu s kolekcí ILink objektů, ze kterých souseda hledáme. Tato třída je v aplikaci použita k hledání podobných popisků.

Třída SimilarByUserWalkLinkGenerationAlgorithm najde další doporučený objekt pro daného uživatele z jeho UserWalks. V konstruktoru se předá DAO k třídě implementující rozhraní IUserWalk a uživatel. V mojí aplikaci to jsou ShopUserWalkDAO, ShopUserWalk a ShopUser. V metodě adapt se předává kolekce walků, ze kterých vybírat.

4.4.1.4 Balíček businesschecks

V tomto balíčku jsou připraveny třídy potřebné pro funkci business check. Ty používám v projektu k zajištění bezchybného chodu DAO. Metody, které potřebují nějakou kontrolu vstupu před uložením do datastore, například kontrola duplikátního jména, tak na svém začátku spustí bussines check. Pokud je vše v pořádku, metoda pokračuje a provedou se změny, ale pokud se vyskytne chyba, je vyhozena BusinessCheckException, která se zpracuje v kontroleru 4.4.3.

4.4.2 Knihovna ASF.Persistence.GAE

V této knihovně se nacházejí abstraktní třídy a rozhraní napsané speciálně pro GAE. Také obsahuje implementace modelů pro GAE.

4.4.2.1 Balíček persistence.gae

Zde je třída abstraktní AbstractPersitable, která zajišťuje, že třídy, které jí rozšiřují, se dají ukládat do GAE datastore.

Pro složitější dotazy a filtry používám v projektu místo DAO query. Rozšíření třídy QueryByJPQL spolu s třídou, která implementuje IQueryRecord, je schopno získávat z datastore odpovědi na libovolné query a následně je i pomocí QueryLazyDataModel z knihovny ASF.Web.Primefaces zobrazit. Query dle předaných parametrů v konstruktoru vytvoří dotaz a odpověď v aplikaci představuje implementace IQueryRecord. Proto parametry, které datastore vrací v odpovědi, se musejí shodovat se strukturou IQueryRecord.

4.4.2.2 Balíček gomawe.storage.gae

Zde jsou implementace modelových tříd pro GAE.

GaeUserModel funguje jako jakási servica. V konstruktoru je jí předán daný uživatel. Metody, které obsahuje, slouží k získávání a ukládání hodnot uložených v entitách GaeUserModelAttribute. Díky této třídě mohu danému uživateli přiřadit k libovolnému objektu parametr a hodnotu.

Třída GaeUserModelAttribute má pět základních parametrů, uchovává v sobě klíč daného uživatele, název třídy entity, ke které je parametr vztažený, a její klíč. Samozřejmě obsahuje také název vlastnosti a její hodnotu.

GaeUserProfile funguje obdobně jako model. V konstruktoru je jí předán daný uživatel. Metody, které obsahuje, slouží k získávání a ukládání hodnot uložených v entitách GaeUserProfileAttribute. Díky této třídě mohu danému uživateli přiřadit libovolný parametr a hodnotu. V aplikaci tuto třídu momentálně nevyužívám, ale v rozšíření by se měla starat o skinovatelnost a další parametry čistě uživatelské.

GaeContentUnitModel funguje podobně jako model. V konstruktoru je jí předán daný objekt. Metody, které obsahuje, slouží k získávání a ukládání hodnot uložených v entitách GaeContentUnitModelAttribute. Díky této třídě mohu danému objektu přiřadit libovolný parametr a hodnotu.

Třída GaeContentUnitModelAttribute má pět základních parametrů, uchovává v sobě název třídy entity, ke které je parametr vztažený, a její klíč. Také obsahuje časový údaj, kdy byl atribut naposledy změněn. Obsahuje také název vlastnosti a její hodnotu.

Všechny implementace DAO rozšiřují AbstractDAO a navíc implementují rozhraní IUserModelAttributeDAO z jádra ASF, které mě donutí implementovat metody potřebné pro práci s atributy.

4.4.3 Knihovna ASF.Web.Primefaces

Tato knihovna v sobě obsahuje správu chyb, AbstractController a QueryLazyDataModel.

AbstractController zajišťuje odchytávání výjimek BusinessCheckException a jejich zobrazení pomocí PrimeFaces Messages. Odchycení se provede pouze pokud je metoda, ve které může vzniknout výjimka, obalena v runnable. Příklad zde:

```
if (tryExecute(new Runnable() {
        @Override
        public void run() {
                commodity = commodityService.create(newName, bd,
                    quant, newDescription);
        }
    })) {
    FacesMessage msg = new FacesMessage("Successful", "Commodity
        "+newName+" created!");
                FacesContext.getCurrentInstance().addMessage(null,
                    msg);
    }
```

Třída QueryLazyDataModel v konstruktoru dostane implementaci AbstractQuery a v generice se jí předá třída implementující daný IQueryRecord. V tuto chvíli ji mohu použít jako PrimeFaces LazyDataModel.

Nakonec zbývá správa chyb a odesílání chybových mailů. Aby se mi chyby správně odchytávaly, je třeba zprovoznit interceptor. Konkrétně to je RuntimeExceptionInterceptor. K jeho zprovoznění jsem musel přidat do projektu AspectJ. Dále je třeba nastavit MailErrorLogger a také mail sender. Logger předám interceptoru a interceptor použiji při vytváření kontroler beanů. Nyní interceptor v provozu a všechny runtime výjimky budou odesílány na zvolený e-mail jako html. Nastavení v applicationContext.xml:

```
<bean name="mailErrorLogger"
          class="cz.cvut.fel.asf.notifications.error.MailErrorLogger">
    <property name="sendAsHtml" value="true"/>
    <property name="emailFrom" value="petr.dobricka@seznam.cz"/>
    <property name="emailTo" value="petr.dobricka@seznam.cz"/>
    <property name="emailSubject" value="Error"/>
</bean>
<bean id="mailSender"
    class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <!-- Specific Google protocol used with Google App Engine -->
    <property name="protocol" value="gm" />
</bean>
<bean name="runtimeExceptionInterceptor"
          class="cz.cvut.fel.asf.tools.error.RuntimeExceptionInterceptor">
    <property name="logger">
        <ref bean="mailErrorLogger" />
    </property>
</bean>
<bean
class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="beanNames" value="*Controller, dataGenerator"/>
    <property name="interceptorNames">
        <list>
            <value>runtimeExceptionInterceptor</value>
        </list>
    </property>
</bean>
```

Odesílání mailů vyžaduje pro převod na html xalan:

```
<dependency>
        <groupId>xalan</groupId>
        <artifactId>serializer</artifactId>
        <version>2.7.1</version>
</dependency>
<dependency>
        <groupId>xalan</groupId>
        <artifactId>xalan</artifactId>
        <version>2.7.1</version>
</dependency>
```

4.4.4 Balíček adaptation v projektu

V tomto balíčku se nachází adaptační třídy, které nepochází z frameworku.

Třída Analyser slouží k adaptaci podle popisků produktů. Pomocí metody TF-IDF ohodnotí tokeny, které získá z popisů produktů. Tím dostanu frekvence, které jsou potřebné pro SimilarByContentLinkAdaptationAlgorithm. V konstruktoru je jí předána pouze CommodityService a analyse počítá se všemi produkty v obchodu a zapisuje do nich frekvence, takže operace je to náročná, a proto jí spouštím pouze pomocí GAE queue.

Třída SimilarBought slouží k adaptaci podle toho, co ostatní koupili také. V konstruktoru jí je předáno GaeUserModelDAO a metoda analyse má parametr Commodity, ke které

hledám ostatní prodané. Metoda najde uživatele, kteří si produkt také zakoupili. Poté najde všechny produkty, které tito uživatelé koupili a spočítá jejich četnost. Za doporučovaný produkt se považuje nejpočetnější z této kolekce a jeho klíč se uloží do atributu modelu produktu poslaného do metody analyse.

ShopUserModel rozšiřuje GaeUserModel a obsahuje metody na převody parametrů mezi obchodem a String hodnotou v GaeUserModelAttribute. V aplikaci používám dvě jména parametrů: bought a rating. První slouží k uchování záznamu, že daný uživatel si koupil daný produkt a hodnota je kolikrát si ho koupil. A rating slouží k uchování záznamu o hodnocení daného produktu daným uživatelem. Hodnota rating představuje počet hvězdiček, takže je od 1 do 5.

ShopContentUnitModel rozšiřuje GaeUserModel a obsahuje metody na převody parametrů podobně jako ShopUserModel. V aplikaci používám pět jmen parametrů. Sold uchovává celkový počet prodejů daného produktu. Rating uchovává průměrné hodnocení daného produktu od všech uživatelů. Ratingstats je pomocný parametr k počítání průměrného hodnocení. Recbought v sobě uchovává klíč doporučeného produktu pomocí SimilarBought. Nakonec recdescription v sobě uchovává dva klíče nejpodobnějších produktů daného produktu.

Třída ShopUserWalk představuje implementaci IUserWalk. Obsahuje list, do kterého se ukládají navštívené produkty. Také uživatele, kterého je to list, session, ve které se uživatel nachází a datum, kdy provedl poslední krok.

Nakonec ShopUserWalkDAOImpl implementuje rozhraní IUserWalkDAO.

Kapitola 5

Testování

Tato kapitola se zabývá otestováním aplikace. Zvolil jsem dva způsoby testování, které zde uvedu.

- Uživateli pomocí scénáře.
- Unit testy.

5.1 Testování aplikace pomocí uživatelů

Ještě před testy s uživateli jsem si aplikaci zkoušel projít sám a vyzkoušel všechny funkce. Nalezl jsem pouze několik problémů se zobrazováním zpráv pomocí PrimeFaces, které jsem ihned opravil. Některé starší části aplikace po úpravách verzí použitých frameworků končily chybou, případně nefungovaly dle požadavků, ale i to po menších změnách šlo opravit. Takto opravenou verzi jsem poskytl k testování pomocí uživatelů.

Aplikaci je dobré otestovat více uživateli, kteří budou mít za úkol projít sestavený scénář skládající se z řady úkolů. Tyto úkoly by měly zahrnovat vyzkoušení funkcí aplikace. Obyčejný uživatel pravděpodobně bude schopen odhalit problémy, které vývojáře nenapadnou. Z odpovědí uživatelů, kteří v plnění úkolů měli nějaké problémy nebo jim něco nebylo jasné, je možné najít kritická místa a upravit je. Toto testování zajistí, že aplikace bude mít co nejintuitivnější ovládání a odhalí chyby v uživatelském rozhraní.

Vytvořil jsem dva scénáře. První se zaměřuje na testování veřejné části D.1 a druhý na testování administrační části D.3. Nejlepší by bylo sehnat skupinu testerů s různou znalostí práce na počítači a věkem. Mně se podařilo sehnat dva uživatele, kteří pracují s internetem občas a tři, kteří s ním pracují denně. Z toho byl jeden do 20let, tři do 30let a jeden nad 50 let. Výsledky veřejné části testů jsou zaznamenány v Tabulce D.1 a administrační části v Tabulce D.2.

Z odpovědí dotazovaných se mi podařilo odhalit chybu, která vznikala při vstupu na detail stránku, pokud daná Commodity neměla žádný obrázek. Problém jsem vyřešil tak, že pokud administrátor nenahraje žádný vlastní obrázek při vytváření nové Commodity, přidám výchozí obrázek ze statických souborů aplikace, ten jde později vyměnit za odpovídající fotku. Dále uživatelé poukázali na problém, že pokud se filtrují Commodity, tak nastavení filtru se ukládá. Přidal jsem proto tlačítko reset, které vrátí všechna nastavení filtru

do výchozí podoby. Chyba s českými znaky vznikala jen ve formuláři, který měl enctype= "multipart/form-data". Pro odstranění chyby jsem ho změnil na enctype="multipart/form-data;charset=UTF-8".

5.2 Testování aplikace unit testy

K otestování logických celků aplikace se dají využít unit testy. Základní sada testů se týká práce se správou dat a servisní vrstvou. Tyto testy ukazují, zda se nově vytvořené nebo změněné entity ukládají do datového úložiště a zda metody a dotazy vrací zpět správná a očekávaná data. Kromě těchto testů jsem ještě připsal testy, které se týkají adaptace. Zde se testují metody sloužící k práci s modelem a UserWalk.

Hlavní testovací třída pro adaptaci se jmenuje AdaptationTest. Zde na uměle vytvořených produktech jsem otestoval základní doporučovací metody. Díky tomuto testu se mi podařilo opravit zásadní chybu v SimilarBought třídě. Bez povšimnutí jsem spatně navrhl metodu, která hledala atributy s danou Commodity. Místo abych hledal jen ty, které mají attributeName "bought", hledal jsem všechny. Proto jsem dodal další metodu do GaeUserModelAttributeDAOImpl, ve které lze zadat attributeName.

Kapitola 6

Závěr

V této práci jsem srovnal několik současných nástrojů pro tvorbu elektronických obchodů. Z toho jsem získal představu, co lze v obchodě adaptovat. Stručně jsem popsal základní charakteristiku Google App Engine a adaptace. Poté jsem navrhl samotnou aplikaci. Využil jsem k tomu Unified Modeling Language. Důležité části obchodu jsem naimplementoval společně s adaptivním chováním pomocí Java Persistance API a frameworků Spring, PrimeFaces a dodaného Adaptive System Framework ve vývojovém prostředí NetBeans. K adaptaci využívám sledování kroků uživatelů, jejich hodnocení produktů, podobnost popisků produktů a zboží, které zákazníci koupili také s vybraným produktem. Na stránce produktu se na základě adaptace pak nabízejí dva nebo tři doporučené produkty. Významnou částí práce je také nahrání aplikace na Google App Engine a následné otestování funkčnosti webu a adaptivního chování. Nakonec jsem se pokusil odstranit co nejvíce chyb, na které se při testování přišlo. V práci jsem také uvedl podrobný návod k nastavení projektu, který bude fungovat na Google App Engine a bude používat Apache Maven a výše zmíněné frameworky. Hlavní cíle této práce se tedy podařilo splnit.

6.1 Přínos a možnosti rozšíření

Přínos této práce vidím hlavně v tom, že v průběhu jsem se nejvíce potýkal s přinucením App Engine spolupracovat se všemi použitými frameworky. Z tohoto důvodu jsem se rozhodl uvést zde návod, jak toho docílit. Samotná práce mi rozšířila obzory v oblasti návrhu aplikace a adaptace, ale hlavně v oblasti implementace webové aplikace. Před začátkem práce jsem neměl žádné zkušenosti s programováním webu pomocí jazyku Java, takže si myslím, že jsem ušel celkem dlouhou cestu. V budoucnu bych mohl na tomto projektu postavit plně funkční obchod.

Možností pro rozšíření tu určitě existují. Mezi největší patří dokončení plně funkčního obchodu s platbami kartou, možnost nahrávání nebo komunikace se zbožím z jiných portálů. V oblasti adaptace jsou tu možnosti skinovatelnosti, sledování dalších statistik (například nejprodávanějšího zboží) a další.

Literatura

[1] Apache maven documentation [online]. ¡http://maven.apache.org/guides/¿, stav z 25.12.2013.

[2] Gae documentation [online]. ¡https://developers.google.com/appengine/docs/java/¿, stav z 25.12.2013.

[3] Gae eclipse jsf návod [online]. ¡http://java.wildstartech.com/Java-Platform-Enterprise-Edition/JavaServer-Faces/javaserver-faces-21¿, stav z 25.12.2013.

[4] Graphical user interface [online]. ¡http://en.wikipedia.org/wiki/Graphical_user_interface¿, stav z 25.12.2013.

[5] M. Çalışkan and O. Varaksin. *PrimeFaces Cookbook*. Packt Publishing, 2013.

[6] J. Arlow and I. Neustadt. *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, 2007.

[7] M. Balík and I. Jelínek. Adaptive system framework: A way to a simple development of adaptive hypermedia systems. 2013.

[8] C. Bauer and G. King. *Java Persistence with Hibernate*. Manning Publications, 2007.

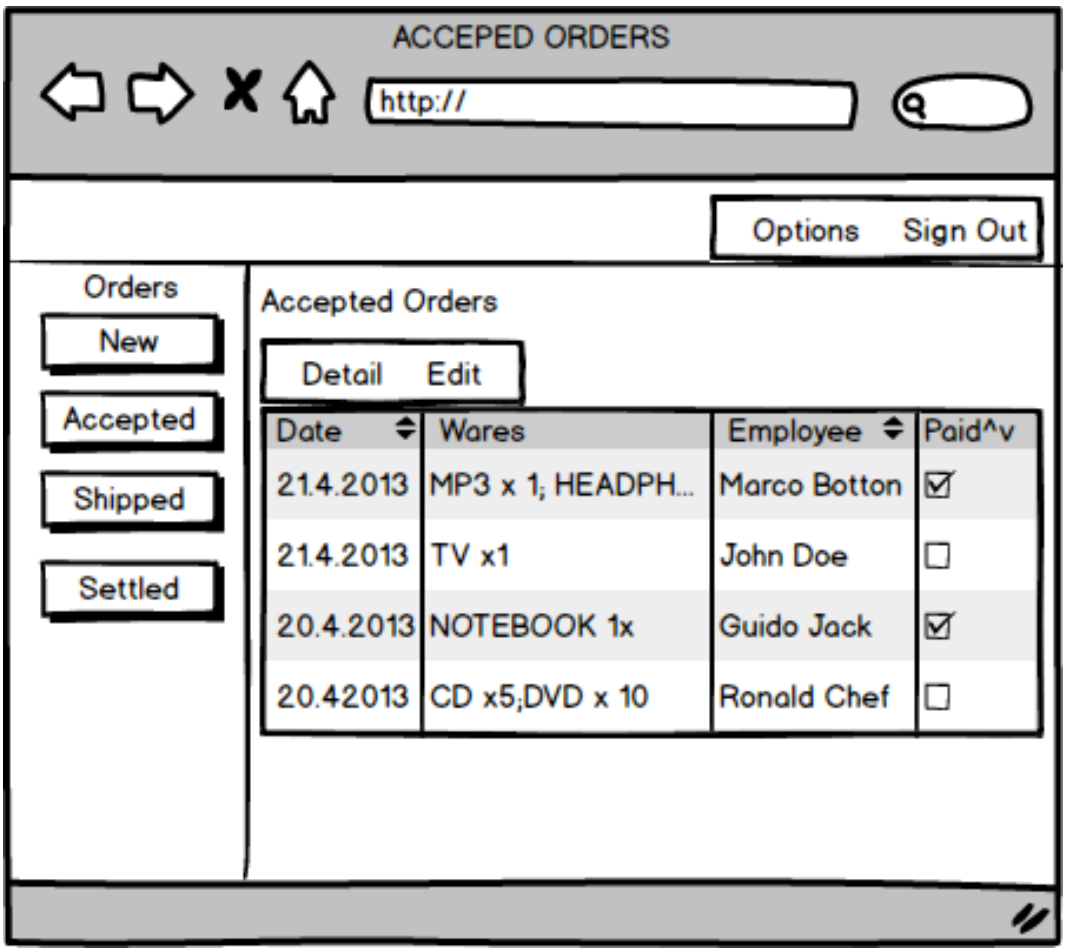
[9] P. Brusilovsky, A. Kobsa, and W. Nejdl. *The Adaptive Web*. Springer, 2007.

[10] D. Sanderson. *Programming Google App Engine*. O'Reilly Media, second edition, 2012.

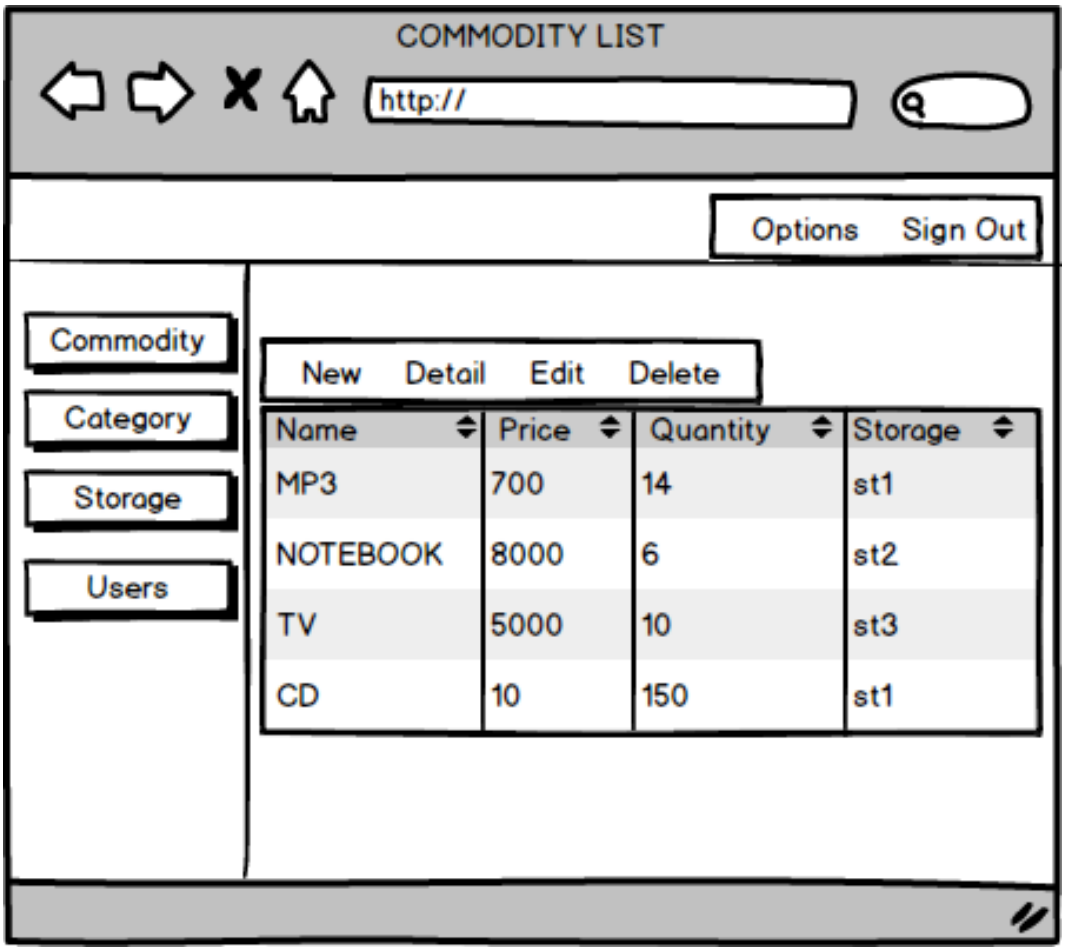
[11] C. Walls. *Spring in Action*. Manning Publications, third edition, 2011.

Příloha A

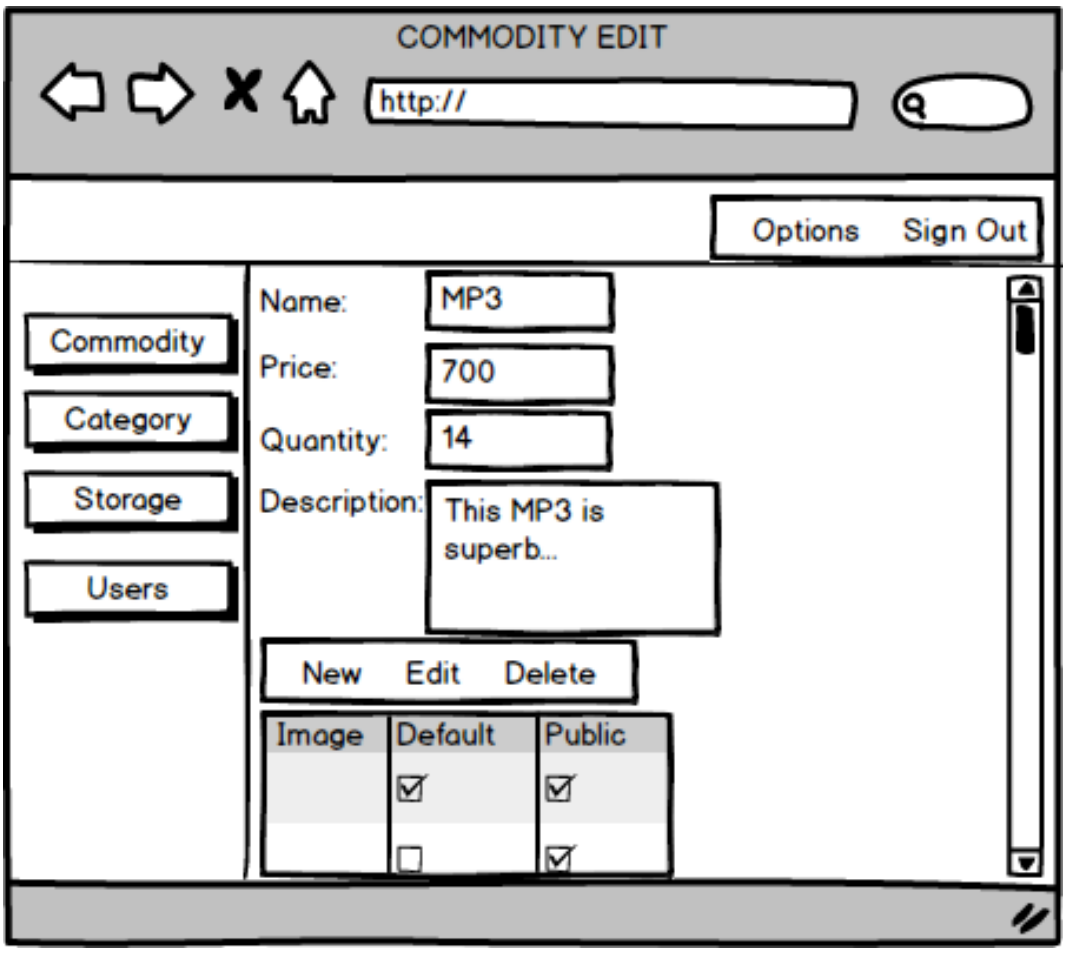
Obrázky



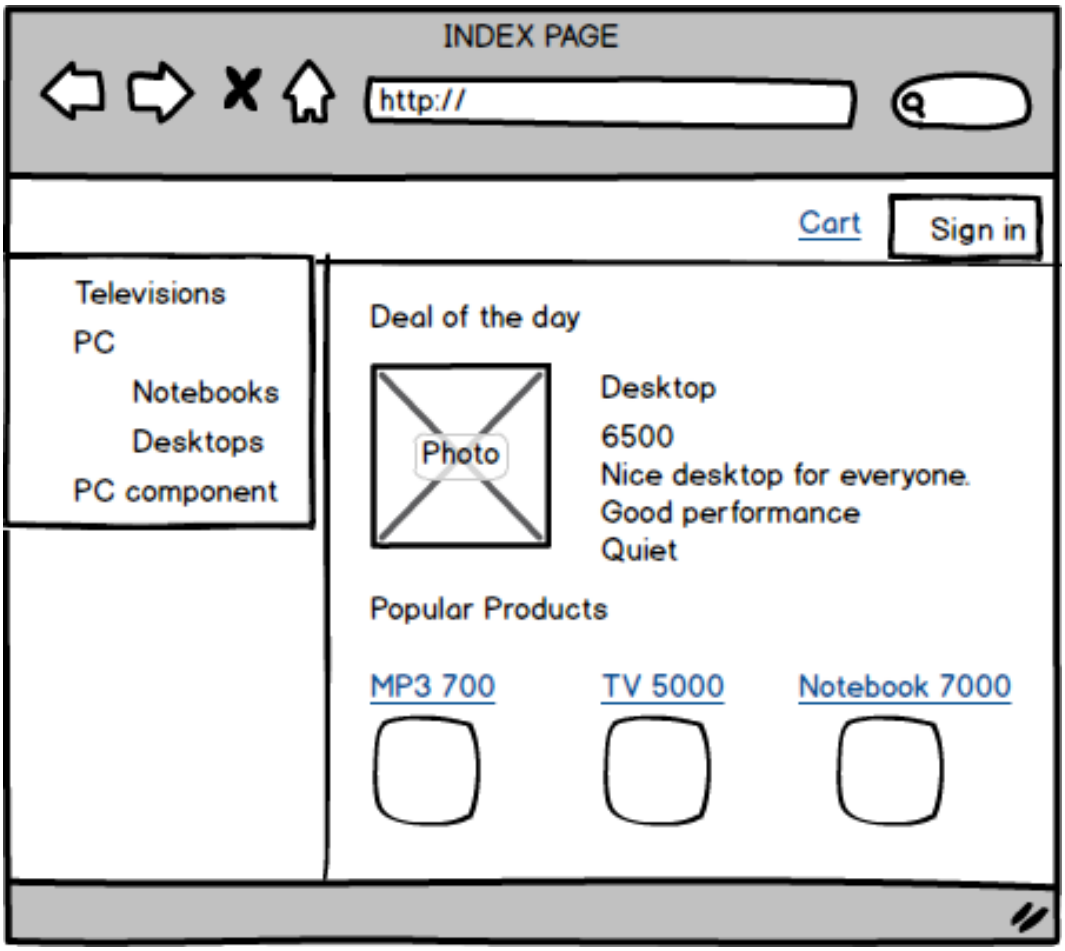
Obrázek A.1: Mockup - přehled přijatých objednávek



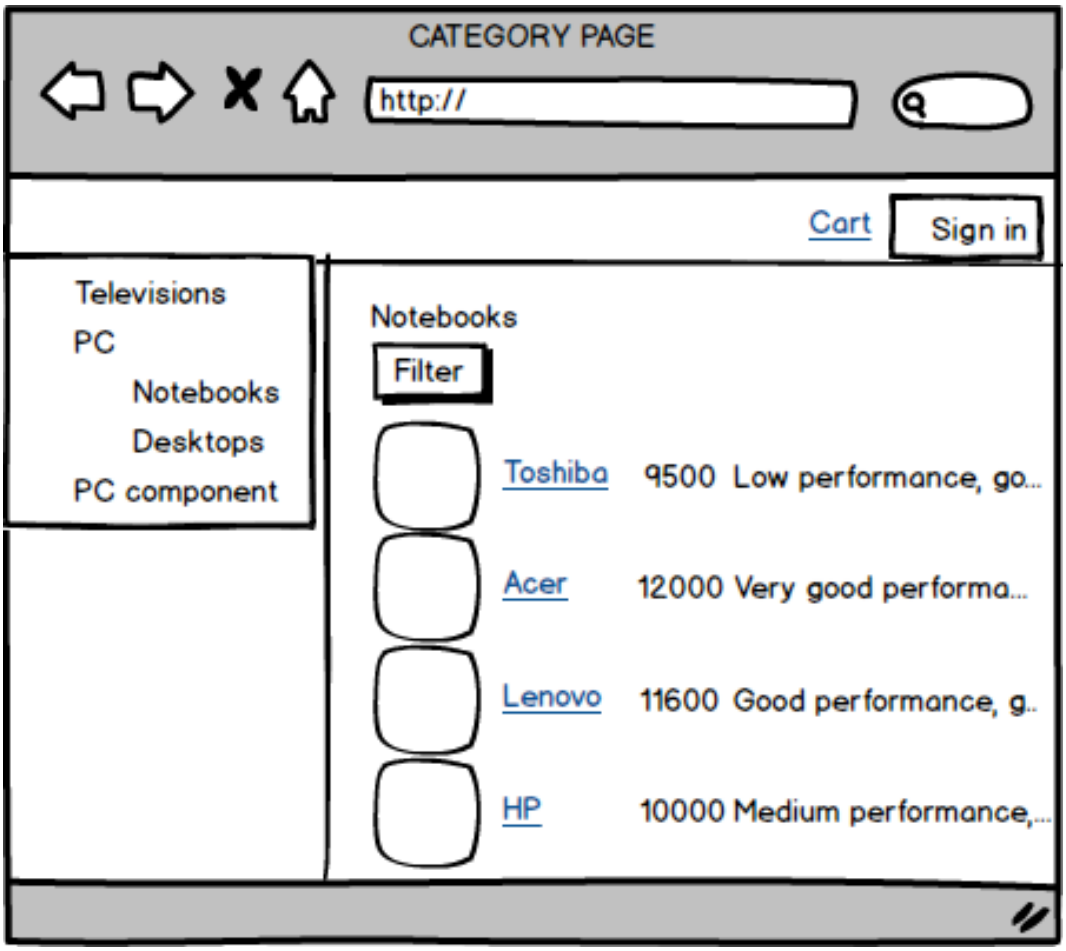
Obrázek A.2: Mockup - přehled produktů - Administrátor



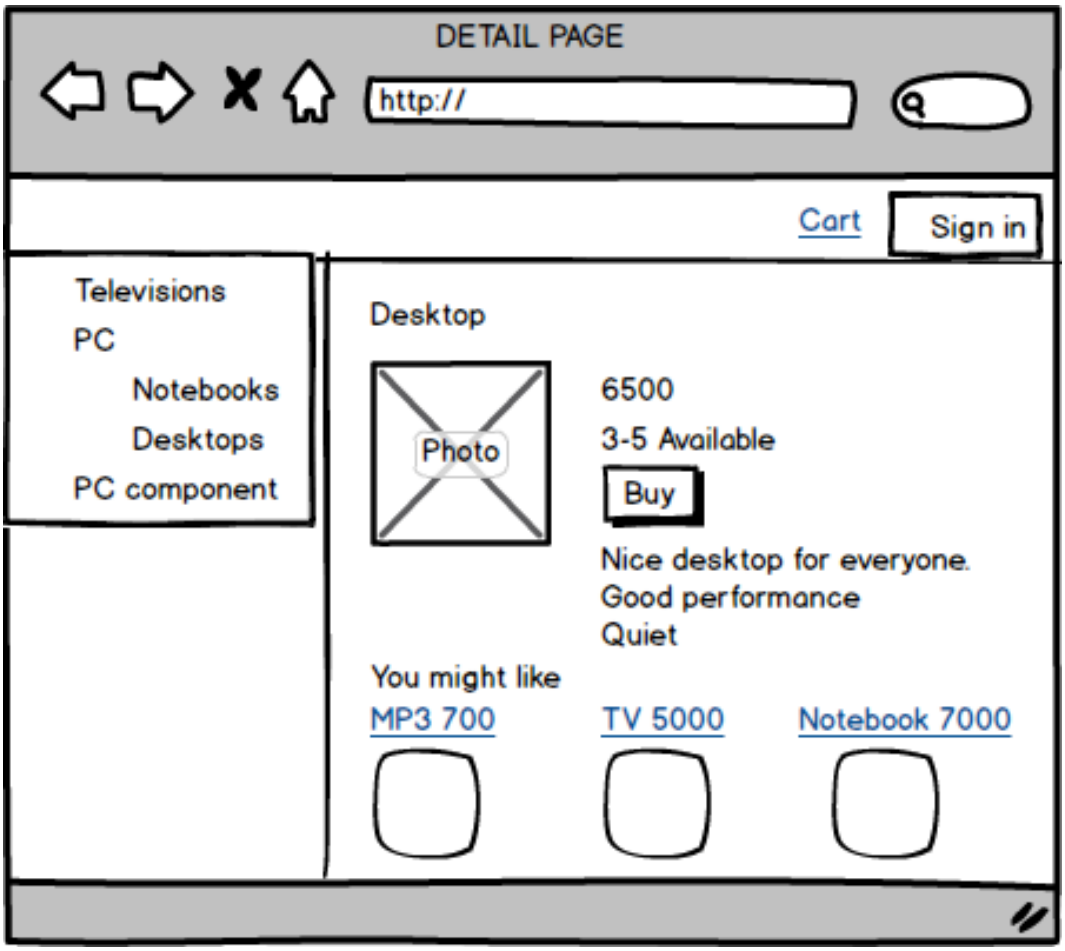
Obrázek A.3: Mockup - editace produktu



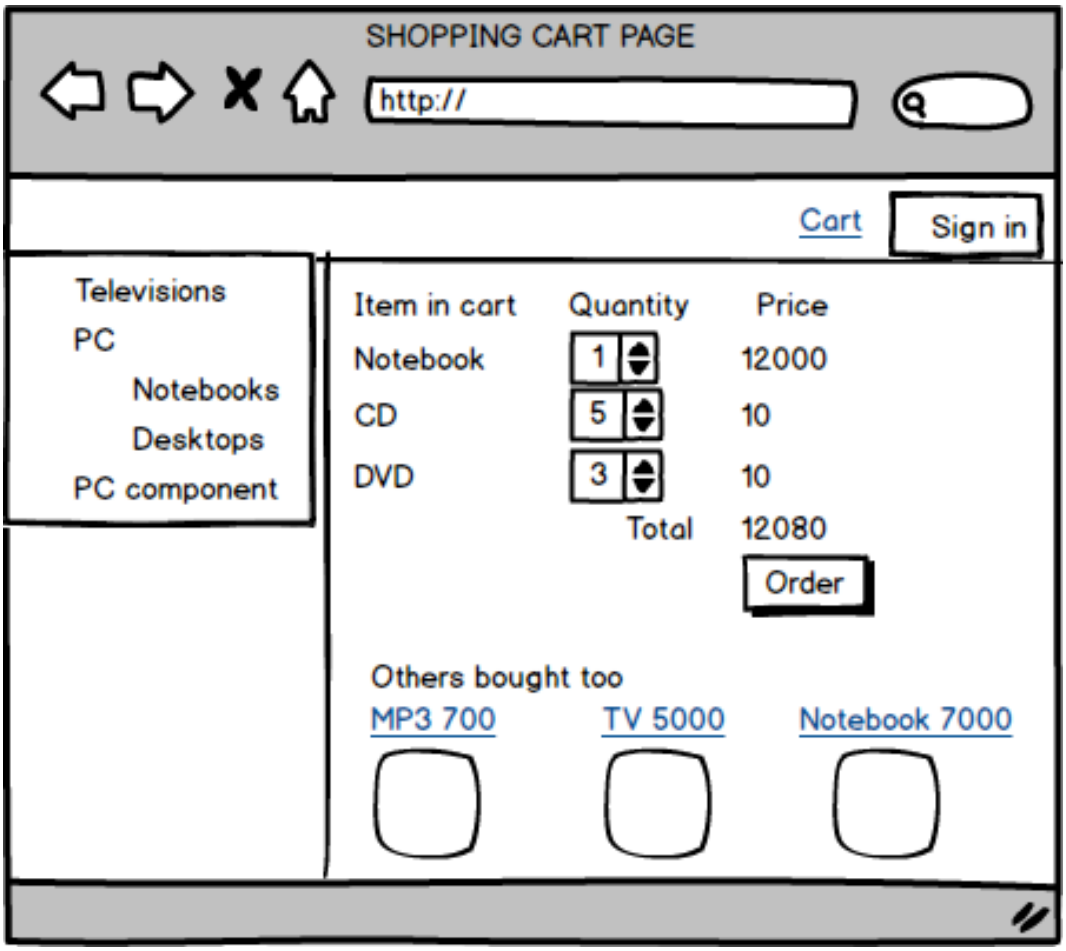
Obrázek A.4: Mockup - úvodní stránka obchodu



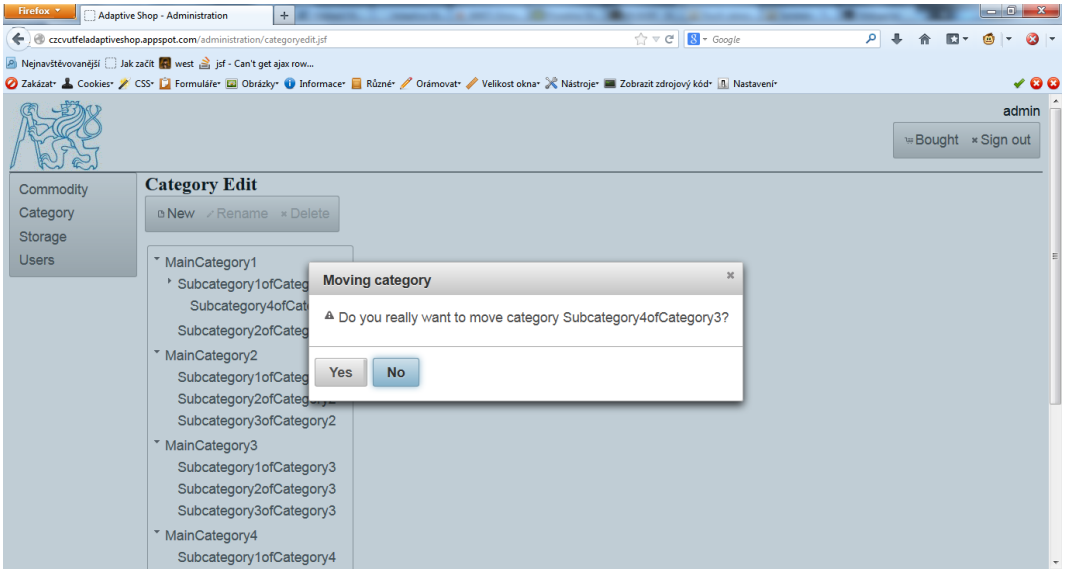
Obrázek A.5: Mockup - stránka s produkty podle kategorií



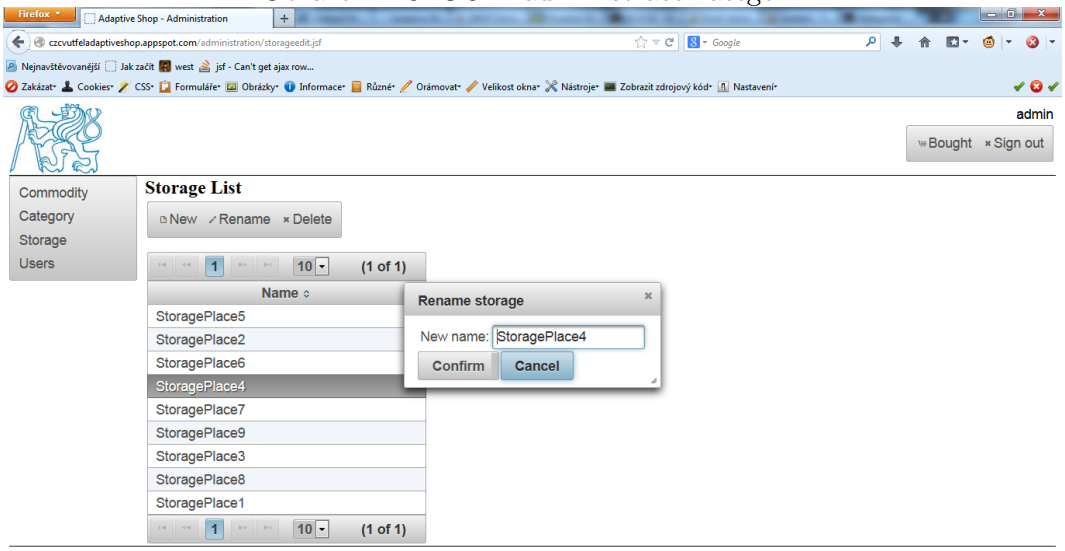
Obrázek A.6: Mockup - stránka s detailem produktu



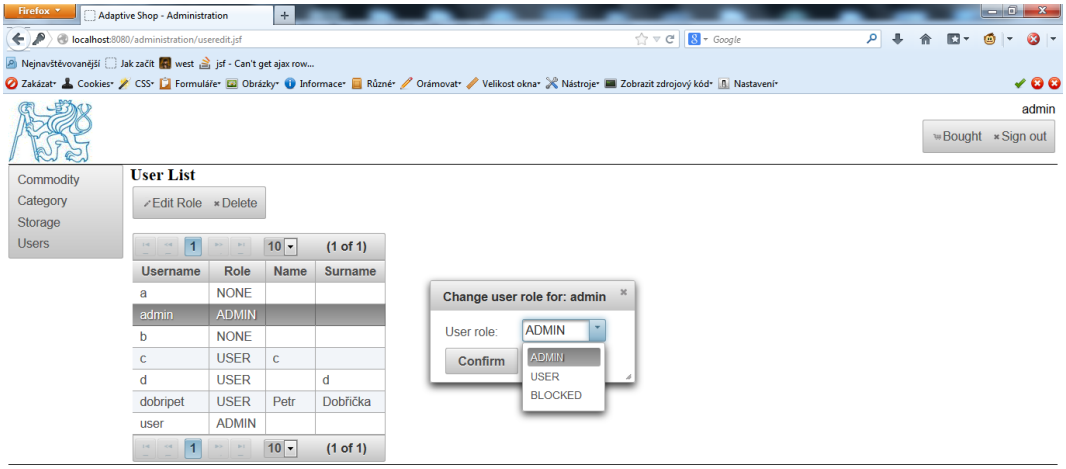
Obrázek A.7: Mockup - stránka s nákupním košíkem



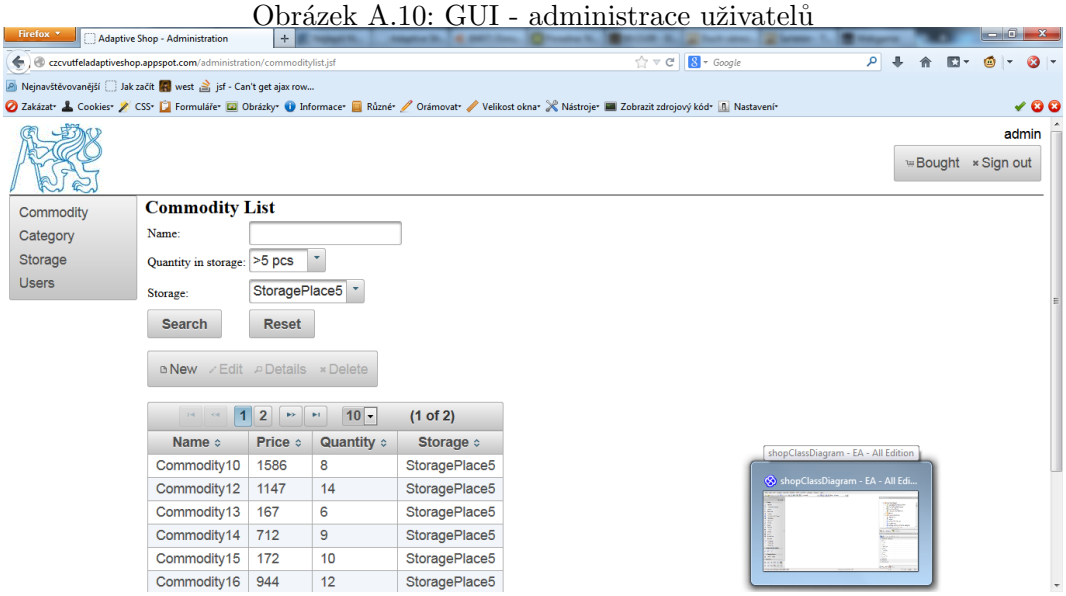
Obrázek A.8: GUI - administrace kategorií



Obrázek A.9: GUI - administrace skladovacích míst

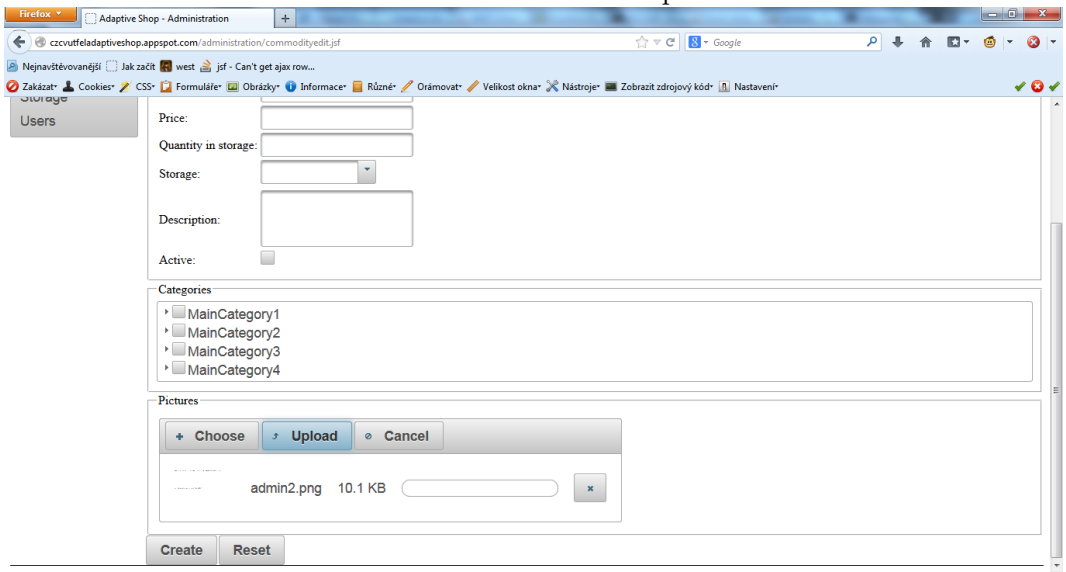


Obrázek A.10: GUI - administrace uživatelů



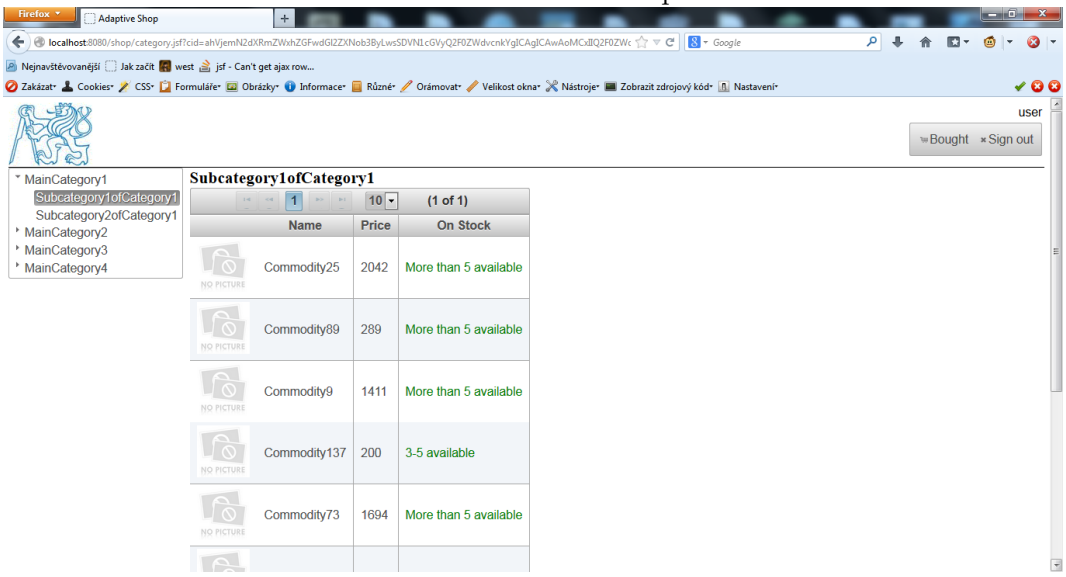
Obrázek A.11: GUI - administrace produktů

Obrázek A.12: GUI - editace produktu

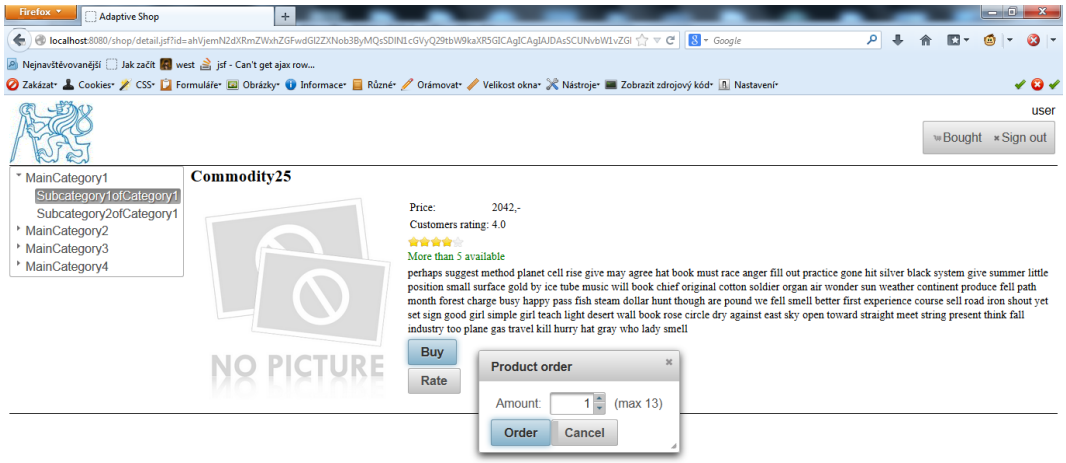


Obrázek A.13: GUI - tvorba nového produktu

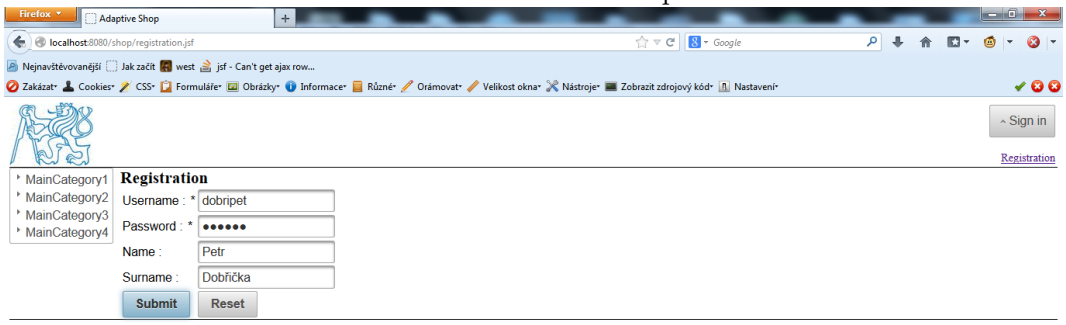
Obrázek A.14: GUI - editace produktu



Obrázek A.15: GUI - tvorba nového produktu



Obrázek A.16: GUI - editace produktu



Obrázek A.17: GUI - tvorba nového produktu

Příloha B

Seznam použitých zkratek

GAE Google App Engine

JDO Java Data Object

JPA Java Persistence API

UML Unified Modeling Language

ASF Adaptive System Framework

JRE Java Runtime Environment

JDK Java Development Kit

JSF JavaServer Faces

GUI Graphical User Interface

SDK Software Development Kit

SQL Structured Query Language

HRD High Replication Datastore

DAO Data Access Object

TF-IDF Term Frequency - Inverse Document Frequency

GOMAWE Generic Ontological Model for Adaptive Web Environments

Příloha C

Instalační a uživatelská příručka

C.1 Instalace potřebných závislostí a programů

Pokus si zvolíte jiné než doporučené cesty nebo jména adresářů, je potřeba podle toho uzpůsobit kroky, ručně extrahovat potřebné soubory a nastavit rozdílně proměnné prostředí.

1. Je potřeba mít nainstalované vývojové prostředí NetBeans[1] s JDK 7[2]. Oboje je na CD ve složce pre-install.

2. Nastavení proměnné prostředí pro Java: JAVA_HOME a PATH.

   (a) Otevřít proměnné prostředí(WinKey+Pause → Upřesnit nastavení systému → Proměnné prostředí).

   (b) V sekci Systémové proměnné vytvořit novou se jménem JAVA_HOME a v hodnotě bude adresa JRE, které je součásti JDK 7 (v mém případě "C:\Program Files\Java\jdk1.7.0_45\jre")

   (c) V sekci Systémové proměnné přidat na konec proměnné PATH adresu složky bin v JDK (v mém případě ";C:\Program Files\Java\jdk1.7.0_45\bin")

3. Pro instalaci Maven 3.1.1 stačí spustit dávkovací soubor install-maven-3.1.1 ze složky pre-install. Nebo ručně:

   (a) Zvolit si místo instalace Maven a tam rozbalit apache-maven-3.1.1-bin.rar ze složky pre-install.

   (b) Nastavit proměnné prostředí pro Maven: M2_HOME a M2 a přidat M2 do PATH.

   (c) Otevřít proměnné prostředí(WinKey+Pause → Upřesnit nastavení systému → Proměnné prostředí).

   (d) V sekci Systémové proměnné vytvořit novou se jménem M2_HOME a v hodnotě bude adresa instalce Maven(v mém případě "C:\Program Files\Apache Software Foundation\apahce-maven-3.1.1")

[1] https://netbeans.org/
[2] http://www.oracle.com/technetwork/java/javase/downloads/index.html

(e) V sekci Systémové proměnné vytvořit novou se jménem M2 a v hodnotě bude "%M2%\bin".

(f) V sekci Systémové proměnné přidat do proměnné PATH ";%M2".

4. Po užití update z ovládání GAE se aplikace nachází na adrese http://czcvutfeladaptiveshop.appspot.com

C.2 Ovládání GAE

Ve složce GAE-commands v projektu se nachází 3 dávkovací soubory, které zajišťují základní operace GAE.

Pro spuštění lokálního testovacího serveru je potřeba spustit GAE-devserver.bat nebo z konzole ve složce projektu příkaz "mvn appengine:devserver". Vypíná se stiskem Ctrl+c a potvrzením. Přistoupit na něj lze z adresy http://localhost:8080/ a administrátorská část, podobná té online na GAE, se nachází na http://localhost:8080/_ah/admin.

Pro nahrání projektu na GAE slouží GAE-update.bat nebo z konzole ve složce projektu příkaz "mvn appengine:update". GAE se zeptá na přihlašovací email a heslo. Pro tento projekt s doménou czcvutfeladaptiveshop je email: petr.dobricka@seznam.cz a heslo: adaptshop13.

Pokud se nahrání nepovede nebo bylo přerušeno, je potřeba spustit opravu. To zajišťuje soubor GAE-rollback.bat nebo z konzole ve složce projektu příkaz "mvn appengine:rollback".

Seznam ostatních operací [3]

Pro vytvoření administračního uživatele na lokálním serveru je potřeba spustit soubor CrateAdmin.bat a zadat '0' pro lokální devserver.

C.3 Registrace nové aplikace na GAE

1. Přihlašte se na https://appengine.google.com/, pokud nemáte Google account vytvořte si nový[4].

2. Zde se nachází seznam Vašich aplikací. Pro registraci nové aplikace pokračujte tlačítkem Create Application.

3. Zde je formulář, ve kterém je potřeba zvolit jméno aplikace a její doménu, také je zde možnost nastavení zabezpečení. V mojí aplikaci se o zabezpečení stará Spring Security, takže je jedno co si zvolím, ale GAE má i vlastní API k rozpoznání uživatelů, kterou lze využívat.

4. Následující stránka už nabízí odkazy pro další správu aplikace. Po kliknutí na dashboard se dostanete už do administrátorského rozhraní aplikace.

5. V tuto chvíli již máte registrovanou doménu a můžete do ní nahrát svojí aplikaci. Stačí aby souhlasil identifikátor nově registrované aplikace s hodnotou v tagu <application>v appengine-web.xml(pro tuto práci je identifikátor czcvutfeladaptiveshop).

[3] https://developers.google.com/appengine/docs/java/tools/maven#app_engine_maven_plugin_goals
[4] https://accounts.google.com/SignUp

C.4 Administrátorská příručka

Předpokládá se, že aplikace je správně nahrána na GAE(Ovládání GAE je uvedeno zde C.2). Pro práci v administrátorském prostředí, je potřeba být přihlášen jako administrátor, proto je potřeba vytvořit administrátorský účet. Ten lze vytvořit spuštěním dodaného souboru CreateAdmin.bat (musí být ve stejné složce s CreateAdmin.jar a složkou lib) z adresáře extras a zvolením '1' pro GAE nebo '0' pro lokální devserver. Po úspěšném přihlášení k GAE můžete zadat libovolné jméno a heslo pro nově vytvářeného administrátora.
Přihlašovací údaje pro tuto aplikace na GAE:

Username petr.dobricka@seznam.cz

Password adaptshop13

Nyní se lze na webu přihlásit pod nově vytvořeným administrátorským účtem a budete automaticky přesměrováni do administrátorské části obchodu. Zde jsou 4 záložky týkající se obsahu obchodu.
Category: Zde jsou všechny kategorie uspořádané do stromu. GUI na Obrázku A.8

- K vytvoření nové kategorie slouží tlačítko New. Stisknutí otevře dialogové okno, kde je třeba zadat jméno kategorie. Po zadání jména lze vytvoření potvrdit tlačítkem Create nebo zavřít toto okno tlačítkem Cancel.
- Tlačítko Delete se zpřístupní po vybrání (označení) kategorie a maže vybranou kategorii. Po kliknutí vyzve k potvrzení této volby, nelze mazat kategorie, které v sobě obsahují jinou kategorii.
- Tlačítko Rename se zpřístupní po vybrání (označení) kategorie a otevře dialogové okno. Po kliknutí vyzve k vyplnění nového jména, tlačítkem Confirm se potvrdí změna, opět lze okno zavřít tlačítkem Cancel.
- Přesouvání kategorií funguje pomocí Drag-Drop, proto stačí kategorii chytnout a přetáhnout na cílové místo. Přesun otevře potvrzovací okénko, kde akce potvrdí stiskem Yes nebo zruší stiskem No.

Commodity: Zde jsou v tabulce zobrazeny produkty. Také je zde jednoduchý filtr pro lepší práci s daty v tabulce. GUI na Obrázku A.11

- Filtr má tři nastavení, která lze kombinovat. Jméno musí mít shodný začátek s jménem produktu, aby filtr správně fungoval. Například pokud máme v obchodu produkt se jménem "Super MP3". Pokud hledáme Super, filtr uspěje. Ale pokud hledáme MP3, tak se v tabulce nacházet nebude. Další jsou roletková menu na výběr místa uložení a počtu produktů na skladě. Nové hledání se provede stiskem tlačítka Search a nastavení filtru se vynuluje tlačítkem Reset.
- K vytvoření nového produktu slouží tlačítko New. Na stránce s formulářem je potřeba vyplnit základní data, to jsou jméno, cena, počet nového zboží v obchodu a popisek. Dále lze vybrat skladovací místo, pomocí checkboxů zvolit, do jakých kategorií zboží

patří a případně nahrát jeho obrázky. Po výběru obrázku je potřeba kliknout na tlačítko Upload před kliknutím na tlačítko Create. Create vytvoří nový produkt, pokud jsou všechny parametry v pořádku. Formulář je možné vyčistit tlačítkem Reset.

- Tlačítko Delete se zpřístupní po vybrání (označení) řádky v tabulce a maže vybraný produkt. Je potřeba tuto volbu potvrdit nebo zrušit v potvrzovacím okénku. Ukázka na Obrázku A.13.
- Tlačítko Edit se zpřístupní po vybrání (označení) řádky v tabulce a otevře stránku s formulářem pro editaci produktu. Jsou zde stejné možnosti, jako u vytváření nového. Změny se potvrdí tlačítkem Confirm. Formulář se vrátí do původního nastavení produktu tlačítkem Reset. Ukázka stránky sloužící k editaci a tvorbě nových produktu je na Obrázku A.12
- Tlačítko Detail se zpřístupní po vybrání (označení) řádky v tabulce a otevře stránku s administrátorským detailem produktu. Jsou zde zobrazeny všechny parametry produktu, které lze v obchodě nastavovat.

Storage: Zde jsou v tabulce zobrazena skladovací místa. Stránka je na Obrázku A.9

- K vytvoření nového skladovacího místa slouží tlačítko New. V dialogovém okně je potřeba zadat nové jméno a potvrdit tlačítkem Create nebo zrušit tlačítkem Cancel .
- Tlačítko Delete se zpřístupní po vybrání (označení) řádky v tabulce a maže vybrané skladovací místo. Je potřeba tuto volbu potvrdit nebo zrušit v potvrzovacím okénku.
- Tlačítko Rename se zpřístupní po vybrání (označení) řádky v tabulce a otevře dialogové okénko, kde lze vybrat nové jméno pro dané skladovací místo. Změnu je třeba uložit stiskem tlačítka Confirm nebo zrušit tlačítkem Cancel.

Users: Zde jsou v tabulce zobrazeni všichni uživatelé s jejich rolemi v systému. Stránka je na Obrázku A.10

- Tlačítko EditRole se zpřístupní po vybrání (označení) řádky v tabulce a slouží ke změně nastavení rolí daného uživatele v systému. Po jeho stisknutí jsou v dialogovém okénku nabídnuty pomocí roletkového menu role v systému. Nastavení se potvrdí tlačítkem Confirm nebo zruší tlačítkem Cancel.
- Tlačítko Delete se zpřístupní po vybrání (označení) řádky v tabulce a maže vybrané uživatele. Je potřeba tuto volbu potvrdit nebo zrušit v potvrzovacím okénku.

C.5 Uživatelská příručka

Obchod se nachází na adrese http://czcvutfeladaptiveshop.appspot.com. Na úvodní stránce jsou nabídnuty tři nejlépe hodnocené produkty obchodu. Úvodní stránka je zobrazena na Obrázku A.14.

V horní části obrazovky lze kliknout na Sign In nebo Registration. Tlačítko Sign In zobrazí stránku s formulářem pro vyplnění přihlašovacích údajů. Pokud jsou správně vyplněny, stiskem tlačítka Login proběhne přihlášení. Tlačítko Cancel slouží k návratu na

domovskou stránku, stejně jako logo obchodu v levém horním rohu stránky. Tlačítko Registration zobrazí stránku s registračním formulářem, pokud jsou údaje validní po stisku tlačítka Submit se vytvoří nový uživatelský účet. Pokud je uživatel přihlášen, v horní části se nachází tlačítko Bought, které zobrazí seznam koupených produktů a tlačítko Sign out, které uživatele odhlásí. Registrační stránka je zobrazena na Obrázku A.17.

Kategorie lze procházet pomocí menu v levé části obrazovky. Pokud je zvolena kategorie, které obsahuje produkty, je zobrazena stránka s produkty dané kategorie. Tyto produkty jsou zobrazeny v tabulce, která obsahuje náhled na obrázek produktu, jeho cenu, jméno a počet na skladu. Na detail produktu lze přejít kliknutím na jeho příslušnou řádku. Ukázka stránky s produkty na Obrázku A.15.

Detailová stránka produktu nabízí jeho cenu, počet na skladu, průměrné hodnocení, popisek, galerii obrázků od všech uživatelů a tlačítka Buy a Rate. Tlačítko Buy je přístupné pouze pokud je v obchodu dané zboží a pro jeho použití musí být uživatel přihlášen. Po kliknutí se zobrazí dialog s výběrem množství ke koupi. Po navolení množství lze koupi potvrdit tlačítkem Order nebo zrušit tlačítkem Cancel. Tlačítko Rate funguje také jen pro přihlášené uživatele. Po jeho stisknutí se zobrazí dialog, ve kterém je možnost ohodnotit produkt hvězdičkami. Hodnocení se potvrdí stiskem tlačítka Confirm nebo zruší tlačítkem Cancel. Detail také obsahuje dva až tři nabídnuté produkty, na které lze přejít kliknutím na jejich obrázek nebo jméno. Ukázka detailové stránky je na Obrázku A.16.

Příloha D

Scénáře testování a výsledky

D.1 Veřejná část - scénář

Do jaké věkové skupiny patříte:

a) <19

b) 20-29

c) 30-39

d) 40-49

e) 50>

Jak často používáte internet:

a) denně

b) občas

c) poprvé

Pokuste se splnit zadané úkoly.

1. V prohlížeči otevřete adresu http://czcvutfeladaptiveshop.appspot.com/shop/index.jsf.
2. Zaregistrujte si vlastní účet a přihlaste se.
3. Kupte si libovolný produkt z libovolné kategorie.
4. Ohodnoťte libovolný produkt.
5. Projděte alespoň další 3 produkty, můžete využít doporučené.
6. Odhlaste se.
7. Přihlaste se. A zopakujte kroky 2,3 a 4.
8. Nakonec si zkontrolujte, zda produkty v seznamu souhlasí s tím, co jste chtěli koupi.

Měli jste potíže s vypracováním nějakého úkolu? Pokud ano, napište jaké.

Co byste zlepšili a naopak co se Vám líbilo?

D.2 Veřejná část - výsledky

Výsledky dotazníků veřejné části jsem zaznamenal do Tabulky D.1.

Věk	**Schopnosti**	**Potíže**	**Hodnocení, návrhy na zlepšení**
a)	a)	Nešla načíst stránka s produkty, po opětovném načtení v pořádku.	V pořádku, ale trochu bych zlepšil rychlost reakcí
b)	b)	Bez problémů.	Bez problému, zlepšení mě nenapadá.
b)	a)	OK	Zlepšil bych, aby v seznamu produktů i malý obrázek fungoval jako odkaz. Jinak OK.
b)	a)	Při vypracovávání úkolů nenastal žádný problém.	Líbí se mi přehlednost a uspořádání informací při otevření detailu jednotlivých produktů. Dále nabídka doporučených produktů na základě mého vlastního výběru. Žádné výrazné zlepšení mě nenapadá, jsem spokojen.
e)	b)	Bez chyb.	Zlepšila bych tabulku se zbožím tak, aby kliknutí na obrázek fungovalo stejně jako kliknutí do řádky.

Tabulka D.1: Přehled výsledků veřejné části

D.3 Administrační část - scénář

Do jaké věkové skupiny patříte:

a) <19

b) 20-29

c) 30-39

d) 40-49

e) 50>

Jak často používáte internet:

a) denně

b) občas

c) poprvé

Pokuste se splnit zadané úkoly.

1. V prohlížeči otevřete adresu http://czcvutfeladaptiveshop.appspot.com/shop/registration.jsf a registrujte se.
2. V prohlížeči otevřete adresu http://czcvutfeladaptiveshop.appspot.com/administration/index.jsf a přihlaste se uživatelským jménem "a"a heslem "a".
3. Vytvořte novou kategorii a libovolně ji přesuňte, přejmenujte.
4. Smažte danou kategorii.
5. Vytvořte nové skladovací místo a přejmenujte ho.
6. Smažte toto skladovací místo.
7. Na dříve vytvořeném účtu si změňte roli a smažte ho.
8. Vytvořte novou komoditu s obrázkem a kategorií.
9. Tuto komoditu najděte pomocí filtru.
10. Přidejte jí další obrázek a změňte ho na defaultní.
11. Původní obrázek smažte.
12. Podívejte se na detail Vaší upravené komodity, zda vše souhlasí.
13. Smažte Vámi vytvořenou komoditu.

Měli jste potíže s vypracováním nějakého úkolu? Pokud ano, napište jaké.

Co byste zlepšili a naopak co se Vám líbilo?

D.4 Administrační část - výsledky

Výsledky dotazníků administrační části jsem zaznamenal do Tabulky D.2.

Věk	**Schopnosti**	**Potíže**	**Hodnocení, návrhy na zlepšení**
a)	a)	Nešlo vybrat uživatele, po opětovném načtení v pořádku.	Celkem v pořádku, jen místy pomalejší.
b)	b)	Nefunguje detail produktu	Až na ten produkt, tak v pohodě.
b)	a)	OK, viz zlepšení	Nepovažuji to za chybu, ale chtělo by zlepšit nastavení filtru, které je stejné i po kliknutí na kategorie a zpět.
b)	a)	Při vypracovávání úkolů nenastal žádný problém.	Vše probíhalo plynule a bez chyb. Systematické řazení je v pořádku a přehledné. Stejně tak i prvky pro výběr možností.
e)	b)	Error při kliknutí na detail zboží, české znaky	Opravila bych chyby, jinak celkem přehledné a pěkné.

Tabulka D.2: Přehled výsledků administrační části

Příloha E

Obsah přiloženého CD

- **/adaptiveshop** - složka hlavního projektu v NetBeans
- **/adaptiveshop/GAE-commands** - složka obsahující soubory pro spouštění GAE příkazů
- **/pre-install** - složka obsahující instalační soubory programů potřebných pro spuštění BP
- **/extras** - složka obsahující CreateAdmin.bat
- **/extras/CreateAdmin** - složka projektu CreateAdmin v NetBeans
- **/text** - složka obsahující PDF verzi této BP