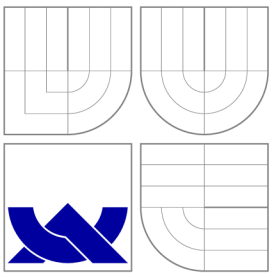
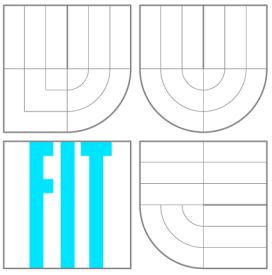


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SROVNÁNÍ NÁSTROJŮ SEAM FORGE A SPRING ROO

THE COMPARISON OF SEAM FORGE AND SPRING ROO TOOLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ NAVLÁČIL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. RADEK KOČÍ, PH.D.

BRNO 2013

Abstrakt

Cílem této bakalářské práce je porovnat nástroje pro rapidní vývoj Spring Roo a JBoss Forge na tvorbě vzorových aplikací, které budou poté nasazeny na aplikační server JBoss AS. Práce v první části popisuje vývoj platformy Java EE, nástrojů a frameworků pro rapidní vývoj internetových aplikací obecně, důvod potřeby těchto nástrojů pro platformu Java EE. Ve druhé části je popsána tvorba jednoduchých aplikací se stejnou funkcionalitou s nástroji Spring Roo a JBoss Forge a porovnání jejich možností, jako podpora tvorby testů, použití na již vytvořené aplikaci, či podpora ve vývojových prostředích.

Abstract

The goal of this bachelor's thesis is to compare the rapid application development tools Spring Roo and JBoss Forge and to deploy created applications on the JBoss AS application server. The first part this thesis describes an evolution of Java EE platform, evolution of tools and frameworks for rapid application development and also reasons for a need of these tools in Java EE platform. The second part describes creation of simple applications with the same functionality using Spring Roo and JBoss Forge. The second part also compares their capabilities in various aspects like testing support, applying these tools on already started projects or support in integrated development environments.

Klíčová slova

Java EE, JBoss AS, JBoss Forge, Spring Roo, rapidní vývoj aplikací, RAD.

Keywords

Java EE, JBoss AS, JBoss Forge, Spring Roo, rapid application development, RAD.

Citace

Ondřej Navláčil: Srovnání nástrojů Seam Forge a Spring Roo, bakalářská práce, Brno, FIT VUT v Brně, 2013

Srovnání nástrojů Seam Forge a Spring Roo

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Kočího, Ph.D. Konzultace mi poskytoval Ing. Karel Piwko. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Ondřej Navláčil

15. května 2013

Poděkování

Děkuji vedoucímu a konzultantovi za vedení, rady a trpělivost, které mi při řešení tohoto projektu poskytli.

Obsah

Kapitola 1

Úvod

Mezi hlavní požadavky kladené na komerční softwarové projekty patří mimo jiné rychlost dodání výrobku na trh, bezpečnost, minimální chybovost společně s kvalitním návrhem a nízkými finančními nároky. Pokrok ve tvorbě softwaru v posledním desetiletí zesílil tlak na rychlost a finanční stránku vývoje produktů. Vzniklé frameworky pro vývoj informačních systémů a internetových aplikací založené na skriptovacích, dynamicky typovaných jazycích, vyvinuly silný tlak na platformu Java a .Net. Za téměř 15 let své existence se Java EE stala standardem používaným miliony vývojářů. Odolat vnějšímu tlaku se snaží uplatňováním moderních architektonických vzorů, moderních principů vývoje software a řadou technologií, které tyto principy efektivně používají.

Úkolem nástrojů jako Spring Roo [3] a JBoss Forge [23] je proces vývoje co nejvíce usnadnit a urychlit. Jejich užití je výhodné zejména v počátečních fázích tvorby aplikace. Nástroje jsou schopny na základě jednoduchých vstupních informací, existující databáze, nebo modelu aplikace, vytvořit konfiguraci projektu společně s objekty, základním uživatelským rozhraním a funkcionalitou. Takto vytvořený kód lze dále upravovat a rozšiřovat s pomocí standardních nástrojů. Na provedené úpravy pak některé nástroje dokáží adekvátně reagovat a provést automaticky změny v příslušných závislých souborech, jež by za normálních okolností musely být provedeny programátorem.

Cílem této práce je porovnat vlastnosti nástrojů Spring Roo a JBoss Forge, porovnání jak teoreticky (použité technologie a principy), tak prakticky (při tvorbě aplikace).

Druhá kapitola práce představí Spring Roo, JBoss Forge a související technologie. Prvních několik odstavců uvede platformu Java EE společně s frameworky Spring [2] a Seam [1]. Dále bude popsán vznik Rapid Application Development (RAD) aplikačních frameworků a budou představeny Spring Roo a JBoss Forge. Poslední podkapitola zmíní aplikační server JBoss AS a nástroj Apache Maven [4].

Třetí kapitola navazuje uvedením základních principů Java EE a RAD ve Spring Roo a JBoss Forge. Principy budou nejprve popsány obecně, pak bude popsáno jejich uplatnění ve srovnávaných nástrojích. Toto povede také k bližšímu seznámení se základy technologií použitých oběma nástroji.

Čtvrtá kapitola nabídne pohled na framework založený na jazyce Python, Django [14]. Jejím cílem je porovnat základní rysy RAD vývoje na platformě Java s jinou platformou.

Pátá kapitola představí ovládání nástrojů Roo a Forge, poté bude srovnána tvorba referenční aplikace. Analýza bude probíhat na dvou jednoduchých informačních systémech se stejnou funkcionalitou vytvořenou s pomocí Spring Roo a následně JBoss Forge. V průběhu tvorby těchto aplikací budou uvedeny

odlišnosti v syntaxi a ergonomii ovládání těchto nástrojů.

Dále budou zpracována témata: podpora v hlavních IDE pro vývoj v jazyce Java, podpora tvorby testů, možnosti využít Spring Roo a JBoss Forge pro úpravu kódu stávající aplikace a odstranění režijního kódu jimi generovaného. Do práce bude zařazeno také srovnání těchto nástrojů v nasazení na JBoss AS aplikačním serveru. V závěru bude představena podpora PaaS (Platform as a Service) přes službu OpenShift společnosti Red Hat.

Kapitola 2

Java EE a RAD nástroje

Jazyk Java je objektově orientovaný jazyk pro obecné použití, vydaný firmou Sun Microsystems v roce 1995. Hlavním důvodem jeho rozšíření bylo od počátku jeho zaměření na oblast tvorby webových aplikací. Nejdříve ve formě jednoduchých appletů, následně komplexních aplikací využívajících jejich komponent. Tento přístup ke tvorbě software vedl k vydání JavaBeans 1.00-A v prosinci roku 1996. JavaBeans definovalo model jazyka Java založený na softwarových komponentách. Tato specifikace představovala množinu pravidel umožňující snadné uspořádání objektů jazyka Java do komplexních celků a jejich znovupoužitelnost, byla ovšem pro tvorbu podnikových aplikací příliš jednoduchá, využití našla převážně při tvorbě uživatelských rozhraní.

2.1 Stručný vývoj technologií Java EE po současnost

Společně se vznikem komplexních rozhraní zaměřených na podnikovou sféru, jako JTA (Java Transaction Api) nebo RMI (Remote Method Invocation) vyvstala potřeba komponentního frameworku, který by sjednotil tyto technologie a vytvořil standard vývojového modelu podnikových aplikací. Úlohu splnil v březnu roku 1998 Enterprise Java Beans (EJB) jako součást kolekce JPE (Java Professional Edition). Specifikace EJB 1.0 přinesla požadované komponenty zaměřené na služby poskytované serverem, ovšem za cenu výrazného navýšení komplexnosti frameworku a nízkého výkonu aplikací.

S příchodem druhé verze Javy v roce 1998 byla platforma rozdělena. J2SE (dnes Java SE) poskytuje základní i pokročilé datové typy a funkcionalitu pro vývoj aplikací a jejich grafického rozhraní, manipulaci s databází, síťovou komunikaci, bezpečnost atd. Její součástí jsou také virtuální stroj a protředky pro vývoj a nasazení aplikací. Platforma pro vývoj podnikových aplikací byla přejmenována na J2EE (dnes Java EE). Definuje rozšíření a běhové prostředí pro tvorbu rozsáhlých, škálovatelných a bezpečných síťových aplikací [25]. J2EE přinesla nové standardy pro bezpečnost, vzdálené volání procedur, prezentační vrstvu, či zefektivnila komunikaci s databázovými technologiemi, ale bez výrazných změn v oblasti produktivity.

Tyto změny přišly v roce 2006 s EJB 3.0 v rámci Java EE 5. Jednalo se zejména o příklon k návrhu řízenému doménami, použití anotací namísto XML konfiguračního souboru (deskriptoru) předchozích verzí a náhrada Entity Beans objekty, které povinně implementovaly rozhraní `EntityBean`, za POJO (plain old Java object) entity. Na konci roku 2009 vyšla Java EE 6. Její významnou novinkou je standard pro vkládání závislostí (Contexts and Dependency Injection for Java, Dependency Injection for Java) [5].

Změny vedly k zjednodušení vývojového modelu a snížily náročnost vývoje v Java EE. Toto se projevilo v lepší přístupnosti platformy nástrojům pro rapidní vývoj aplikací.

Java	Java SE	Java EE	Rok
JDK 1.0	X	X	1996
JDK 1.1	X	X	1997
X	J2SE 1.2	JPE	1998
X	X	J2EE 1.2	1999
X	J2SE 1.3	X	2000
X	X	J2EE 1.3	2001
X	J2SE 1.4	X	2002
X	X	J2EE 1.4	2003
X	J2SE 5.0	X	2004
X	Java SE 6	Java EE 5	2006
X	X	Java EE 6	2009
X	Java SE 7	X	2011

Tabulka 2.1: Historie verzí jazyka Java

2.1.1 Seam Framework

Seam Framework [1] byl vytvořen společností Red Hat za účelem poskytnout plně integrovanou vývojovou platformu pro tvorbu internetových aplikací odpovídajících standardu Java EE, schopné tradičního i cloudového nasazení. Seam slouží jako prostředek rozšiřující zejména technologie Java Server Faces a EJB. Pro účely perzistence dat pak standardně využívá JPA a Hibernate. V jeho poslední, třetí, verzi je tvořen kolekcí modulů a nástrojů založenou na standardu CDI (Context and Dependency Injection) [5]. Implementace tohoto standardu firmou Red Hat se nazývá Weld [6]. Důležitou vlastností vycházející z CDI je modularita a přenositelnost takovýchto modulů mezi prostředími pro běh aplikací standardu Java EE, aplikačními servery a servletovými kontejnery. Jako doplněk pro urychlení vývoje Seam 3 aplikací byl vytvořen Seam Forge (v roce 2012 přejmenován na JBoss Forge). Verze JBoss Forge 1.0 byla vydána v únoru roku 2012. Dalším nástrojem pro usnadnění vývoje za použití Seam frameworku jsou kromě JBoss Forge i JBoss Tools, modul pro vývojové prostředí Eclipse, integrující technologie pro vývoj Seam aplikací včetně aplikačního serveru JBoss AS a balíčkovacího nástroje Apache Maven.

V nedávné době byly práce na Seam Frameworku zastaveny. Weld a některé další jeho součásti daly vzniknout novému projektu Apache DeltaSpike, jiné (například JBoss Forge) se staly samostatnými projekty.

2.1.2 Spring Framework

Základ Spring frameworku [2] byl vytvořen Rodem Johnsonem a představen v jeho knize Expert One-on-One: J2EE Design and Development [26] vydané v roce 2002. Představen byl jako "interface 21

framework". Jeho cílem bylo vytvoření open-source standardu pro vývoj Java EE aplikací, který by dokázal zastoupit zejména EJB před verzí 3.0. Umožňuje budovat aplikace z jednoduchých objektů (plain old Java objects) a aplikovat na ně podnikové služby neinvazivním, vývojáře neomezujícím, způsobem s využitím vkládání závislostí a aspektově orientovaného programování. Spring je možné použít jak ke tvorbě aplikací na platformě Java SE, tak Java EE.

Verze 1.0 vyšla na jaře roku 2004. V současnosti je aktuální verze 3.2. Jeho vývoj vede společnost Springsource, dnes již jako divize společnosti VMWare. Společně s ní firma Springsource vyvíjí přidružené technologie jako Spring Security, Spring Web Flow [2], či Spring Roo a další. Spring Roo bylo ve své první verzi představeno v prosinci roku 2009.

2.2 Příchod RAD nástrojů

Původním významem pojmu rapid application development je proces tvorby softwaru zaměřeného na iterativní vývoj a tvorbu prototypů. Dnes se používá i v kontextu nástrojů urychlujících vývoj [12]. Nástroje pro rapidní vývoj aplikací tvoří abstraktní vrstvu nad mateřským prostředím, či jazykem. Tato vrstva umožňuje nakládat s prostředky, tvořit kód aplikací a konfigurovat prostředí zjednodušeným způsobem.

Současný trend open source technologií umožňujících rapidní vývoj internetových aplikací začal v roce 2004 s uvolněním kódu frameworku Ruby on Rails jeho tvůrcem Davidem Heinemeierem Hanssonem. Původním účelem frameworku bylo zjednodušení vývoje komerční internetové aplikace pro správu projektů. Rails staví na metaprogramování, vhodných návrhových vzorech a principech softwarového inženýrství, jako Scaffolding, Active Record, Model View Controller (MVC), Convention over Configuration, Don't repeat yourself (budou později rozvedeny). Framework využívá skriptovacího jazyka Ruby. Příkazy uživatele předané z příkazové řádky generují počáteční strukturu a konfiguraci projektu, entity a proměnné, základ uživatelského rozhraní, nebo spouští testovací server. Pokročilé úpravy vykonává programátor standardní cestou v editoru. Pro prezentaci slouží CSS, HTML a Javascript. Jako databáze je zpravidla využito MySQL, či PostgreSQL, aplikace je nasazena na Apache HTTP Server. Úspěch Rails postupně vedl ke vzniku mnoha dalších frameworků podobné koncepce, založených na jiných skriptovacích jazycích, například Django v jazyce Python, nebo CakePHP v jazyce PHP atd.

Příchod vysoce produktivních frameworků založených na skriptovacích jazycích znamenal růst jejich popularity a zesílený tlak na platformu Java. Vznikl proto například dynamicky typovaný skriptovací jazyk určený čistě pro JVM, Groovy. Vývoj tohoto jazyka započal v roce 2003, dnes je aktuální verze 2.0. Jeho základní vlastnosti a filozofie jsou inspirovány Pythonem, Ruby, či Smalltalkem. Poskytuje kompatibilitu s knihovnami a kódem vytvořenými v Javě, svou syntaxí je Javě blízký. Na Groovy staví framework Grails (dříve Groovy on Grails), nadstavba Spring pro rapidní vývoj aplikací. Alternativou jsou implementace interpretů oblíbených skriptovacích jazyků, určené pro běh na virtuálním stroji jazyka Java, například JRuby (Ruby) a Jython (Python) [27]. Oba podporují tvorbu desktropových aplikací i internetových aplikací ve frameworcích původního jazyka i Javy EE. Podobně jako u Groovy jsou zdrojové soubory těchto jazyků kompilovány do Java bytekódu. Jejich kód může být volán z programů v Javě a naopak metody a třídy Javy mohou být volány, nebo instanciovány z programu v Groovy, Jythonu, nebo JRuby. Uvedené projekty jsou sice plně konkurenceschopné, ale někteří vývojáři mohou preferovat řešení založené na čistém jazyce Java a platformě Java EE. Blízko ke splnění těchto požadavků má jazyk

Groovy a framework Grails, ještě blíže potom nástroje jako Spring Roo a JBoss Forge.

2.3 Spring Roo a JBoss Forge

Spring Roo a JBoss Forge lze definovat jako jednoduše použitelné, produktivní nástroje pro tvorbu podnikových aplikací v jazyce Java. Oba komunikují s uživatelem skrze vlastní rozhraní v příkazové řádce, výhradně formou textu. Jsou spustitelné samostatně v prostředí operačního systému, případně ve vývojovém prostředí ve formě doplňku (Roo) [3]. Využívané jsou pouze při vývoji aplikace a nijak se nepodílí na jejím chodu po nasazení. Specifické znaky jejich používání (v případě Spring Roo AspectJ soubory) jsou snadno odstranitelné, nebo žádné. Dalším významným znakem je modularita těchto nástrojů. Do obou lze nainstalovat moduly rozšiřující možnosti, co se týče pestrosti použitých technologií i ovládání.

Vytvořená aplikace představuje jednoduchý MVC program (prototyp), umožnující provádět nad modelem programu základní operace. Tato kostra má strukturu projektu vytvořeného nástrojem pro tvorbu verzí a kompilaci aplikací Apache Maven (viz kapitola 2.6). Z příkazové řádky JBoss Forge a Spring Roo lze Apache Maven používat.

2.4 Nasazení aplikace

Webová aplikace v Java EE je zpravidla nasazena na aplikačním serveru, nebo servletovém kontejneru. Server aplikacím poskytuje prostředky zahrnující služby datové, transakční, bezpečnostní, správu rozsáhlých distribuovaných systémů, či kontrolu zátěže. V této práci je využíván aplikační server JBoss AS. Nasazení korektně vytvořené a sestavené aplikace na samostatně běžící server JBoss AS spočívá pouze ve vložení balíčku podporovaného typu JAR archivu (především WAR a EAR viz 2.5) do složky deployments v domovském adresáři serveru. Tato složka je skenována, nasazené aplikace jsou dynamicky spravovány. Nasazení je možné i skrze plugin, například buildovacího (build znamená sestavení aplikace.) nástroje Apache Maven, či prostředí Eclipse. JBoss AS poslední verze (7.1.x) splňuje specifikaci Java EE 6 full profile i web profile, což znamená plnou podporu posledních Java EE technologií.

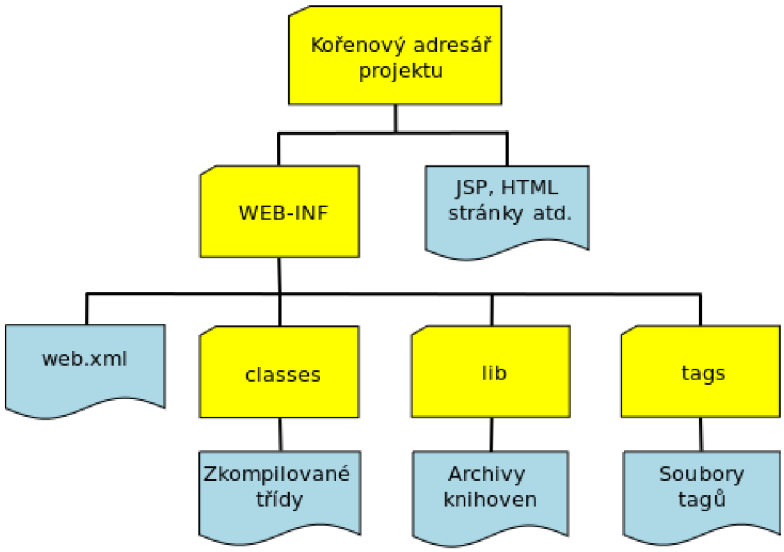
Servletový kontejner v porovnání s aplikačním serverem neobsahuje ve svém základu plnou podporu Java EE specifikace, ale pouze základní funkcionalitu pro běh internetové aplikace (podpora servletů, prezentační a perzistenční vrstvy).

Servletem rozumíme třídu jazyka Java rozšiřující služby serveru poskytované běžícím aplikacím. Jeho úkolem je zpracovávat přicházející požadavky a odpovídat na ně.

2.5 Archiv JAR, WAR, EAR

JAR (Java Archive) je archiv formátu ZIP, nesoucí třídy jazyka Java a další dodatečné soubory (například konfiguraci, obrázky, atd.). Soubory typu **.jar* slouží jako stavební bloky Java aplikací [11].

Jelikož balíčky s aplikacemi používané v práci jsou typu WAR [16], následuje detailnější popis vlastností tohoto archivu. WAR (Web application ARchive) je používán pro distribuci webových aplikací.



Obrázek 2.1: Struktura archivu WAR

Obsahem balíčku bývají Java servlety, JSP soubory, HTML soubory, obrázky a další soubory tvořící aplikaci. WAR má specifickou strukturu. Aplikace uvnitř něj obsahuje adresáře *WEB-INF* a nepovinný *META-INF*.

- Součástí adresáře *META-INF* mohou být soubor *manifest.mf* obsahující informace o souborech archivu, dále pak digitální podpis tvůrce, soubor *index.list* s informacemi o umístění souborů a adresář services s konfigurací používaných služeb. Obsah tohoto adresáře je vytvářen automaticky (pomocí nástroje Ant nebo Apache Maven).
- *WEB-INF* především obsahuje deskriptor (deployment descriptor) *web.xml* [16]. Účelem *tohoto souboru* je zejména definice mapování služeb serveru, které aplikace používá, na URL. Dále v adresáři *WEB-INF* můžeme nalézt podadresáře *classes*, nesoucí zkompilované třídy aplikace, *lib* s archivy použitých knihoven, *tags* nesoucí fragmenty JavaServer Pages kódu použitelné ve formě tagů. Obsah *WEB-INF* se liší v závislosti na použitých technologiích a komplexnosti aplikace.

Archiv EAR (Enterprise Application Archive) je určen k nasazení na aplikačním serveru. Nese podnikovou aplikaci, jejíž součástí jsou zpravidla i prvky Java EE nepodporované servletovým kontejnerem (např. EJB). Archiv je složen z modulů (nejčastěji ve formátu WAR a JAR) a konfigurace v adresáři *META-INF*.

2.6 Nástroj Apache Maven

Apache Maven [4] má za úkol automatizované vykonávání při vývoji často prováděných akcí. Jeho účelem je zvýšit efektivitu práce. Ovládán je v příkazové řádce, nebo jako plugin ve vývojovém prostředí. Mezi jím podporované akce patří například kompilace zdrojového kódu do binárního, tvorba balíčků z binárních souborů, spouštění testů, nasazení aplikací, tvorba dokumentace k jednotlivým verzím software a zejména správa závislostí.

Hlavní součástí každého projektu využívajícího Apache Maven je soubor *pom.xml* (Project Object Model, zkráceně POM) v kořenovém adresáři. Jeho obsahem jsou metadata reprezentující samotný projekt,

jako konfigurace projektu, jeho organizace, nastavení závislostí a další. Apache Maven projekty mají také specifickou adresářovou strukturu (viz struktura vytvořených aplikací v příloze C a D). Tento jednotný způsob konfigurace společně se systémem správy závislostí vede k snadné reprodukovatelnosti projektu. POM je účinný při i tvorbě projektů složených z mnoha modulů. Jeho použití napomáhá zpřehlednění struktury projektu, práci v týmech a usnadňuje změny funkcionality uchovávané v modulech.

Kapitola 3

Důležité principy využívané RAD frameworky a nástroji

Jak bylo zmíněno v kapitole 2.2, framework Ruby on Rails využívá významné návrhové vzory a principy softwarového inženýrství. Tyto jsou platné i pro Spring Roo a JBoss Forge. Nadcházející kapitola je představí a přiblíží jejich využití v porovnávaných nástrojích, společně s nimi bude popsáno i vkládání závislostí a aspektově orientované programování.

3.1 Inverze kontroly

Inverze kontroly (IoC) [2] je jedna ze základních vlastností Java EE 6 a Spring. V objektovém programování bývají závislosti mezi jednotlivými objekty utvářeny statickým přiřazováním jednoho objektu druhému. V komplexních programech toto klade vysoké nároky na vhodný návrh aplikace, organizaci objektů do funkčního celku. Jako částečné řešení se nabízí v podobě návrhových vzorů, které je však nutné opětovně implementovat. Inverze kontroly ve Spring Frameworku i Java EE 6 nabízí formální způsob, jak uspořádat funkční aplikaci z různorodých komponent. Toto spočívá v předání některých vlastností náležících vytvářenému objektu do rukou použitého IoC frameworku. Může se jednat například o vytváření nových objektů, předávání inicializovaných objektů, doplňování objektů o nové funkce. V rámci Javy se inverze kontroly uplatňuje při vkládání závislostí (Dependency Injection) [2] a aspektově orientovaném programování (Aspect Oriented Programming) [2]. Základem IoC frameworku je IoC kontejner, který spravuje objekty a vkládá deklarované závislosti.

3.1.1 Vkládání závislostí ve Spring Framework a Java EE 6

Vkládání závislostí ve Spring Framework je proces, kdy objekty definují své závislosti (další objekty, se kterými pracují) například v argumentech konstruktoru, argumentech factory metody, nebo v proměnné. Prostředí (IoC kontejner) vloží objekt deklarovaný v závislosti (v konfiguračním XML souboru či anotaci), jakmile je tento objekt vytvořen. Vložení závislosti uvozují ekvivalentní anotace `@Autowired`, nebo `@Inject`. Druhá jmenovaná je definovaná specifikací Java EE 6, Spring ji podporuje od verze 3.0. Použití anotací vede k přehlednějšímu kódu a snadnější aplikaci testů na vytvořené třídy. Vkládání závislostí

ApplicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
<bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
</bean>
<bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
</bean>
    <!-- more bean definitions go here -->
</beans>
```

Obrázek 3.1: Konfigurace ApplicationContext.xml

SimpleMovieListerSpring.java

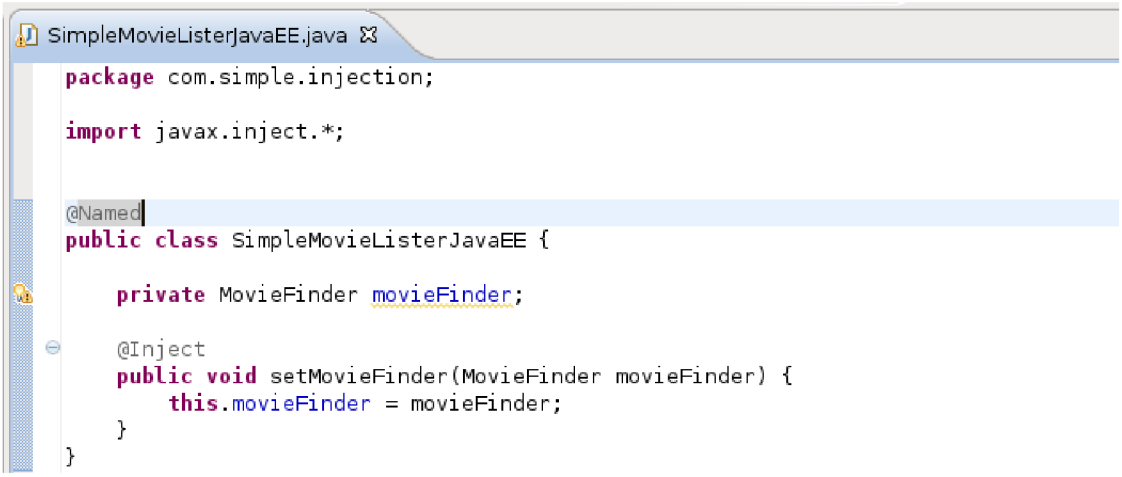
```java
package com.simple.injection;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class SimpleMovieListerSpring {
    private MovieFinder movieFinder;

    @Autowired
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

}
```

Obrázek 3.2: Registrace objektu a příklad vložení závislosti ve Spring

má na starost rozhraní ApplicationContext, které v aplikaci reprezentuje IoC kontejner. Jeho vlastnosti lze nastavit ve stejnojmenném XML souboru, nebo pomocí anotací. V závislosti na konfiguraci jsou vytvořeny instance deklarovaných tříd a instance IoC kontejneru Spring [2].

V příkladu 3.1 je šablona souboru *ApplicationContext.xml* s registrací bean objektů spravovaných IoC kontejnerem. Na obrázku 3.2 je příklad registrace objektu ve Spring anotací `@Component` a vložení registrovaného objektu do hlavičky metody anotací `@Autowired`.

Vkládání závislostí v Java EE 6 je uskutečněno za pomocí standardu CDI (Context and Dependency Injection) [5]. Z pohledu CDI je vkládání závislostí typově bezpečným způsobem, jak vložit do aplikace komponenty s možností zvolit při nasazení konkrétní implementaci vkládaného rohraní. Java EE 6 společně s CDI představila Managed Beans. Jedná se o jakýkoliv objekt POJO obsahující prázdný konstruktor, nebo konstruktor označený anotací `@Inject`, včetně EJB objektů. Takový objekt je po vložení prázdného souboru *beans.xml* do složky WEB-INF (u webových modulů) nebo META-INF (u JAR modulů) automaticky spravovaný kontejnerem. Pomocí anotace `@Inject` je uvozen vkládaný objekt. Na obrázku 3.3 je příklad anotace `@Named`, která z objektu vytvoří Managed Bean a zpřístupní jej pro použití



Obrázek 3.3: Registrace objektu a příklad vložení závislosti v Java EE 6

v prezentační vrstvě aplikace. Na témže obrázku je příklad vložení závislosti (třídy MovieFinder) anotací `@Inject`. Tento způsob je podporován i ve Spring framework.

Framework JSF (viz kapitola 3.2.2), používaný v základním nastavení JBoss Forge, nabízí i vlastní implementaci vkládání závislostí. Spravovaný a vkládaný objekt deklarují anotace `@ManagedBean` a `@ManagedProperty`. Rozdíly mezi tímto způsobem vkládání závislostí a CDI vyplývají ze zaměření frameworku JSF na tvorbu webových aplikací. Tento přístup nepodporuje kompletní specifikaci Java EE 6, nabízí tedy podmnožinu funkcionality v porovnání s CDI. Například zde není podporováno vkládání kontejnerem spravovaných EJB objektů a Aspektově orientované programování standardu Java EE. [5] [6].

3.1.2 Aspektově orientované programování

Existuje mnoho frameworků umožňujících aspektově orientované programování (AOP) v jazyce Java. Patří mezi ně i Spring framework a AspectJ, používaný nástrojem Spring Roo. AOP podporuje také Java EE 6. Aspekt je vlastnost s uplatněním v rámci celé aplikace. Může se jednat například o kód spravující bezpečnost, logování, nebo transakce. Aspektový framework dokáže přidat takovéto vlastnosti objektům za běhu aplikace, nebo při její kompilaci. Součástí AOP je několik důležitých konceptů.

- Join point je moment vykonávání operace, například volání metody či výjimky. Zde se uplatní operace definované AOP kódem (Advice).
- Advice je zpravidla formou interceptoru, návrhového vzoru a rozhraní jazyka Java používaného ke změně toku aplikace a vložení nového chování v místě Join pointu.
- Jako Pointcut je označován predikát popisující případy, kdy má být kód aspektu uplatněn [2]. V Join pointech, které vyhovují jeho pravidlům, je vykonáno příslušné Advice.

Advice dělíme na několik druhů podle místa jejich vykonání vzhledem k Join pointu.

- Before advice se vykoná před Join pointem bez možnosti zabránit jeho vykonání jinak, než výjimkou.
- After advice může být vykonáno po úspěšném ukončení Join pointu (After returning), po vyvolání výjimky při jeho výkonu (After throwing), nebo bez ohledu na způsob ukončení (After finally).
- Around advice, pak lze uplatnit před i po Join pointu, dokáže zabránit jeho vykonání a vrátit hodnotu za něj, případně vyvolat výjimku podobně jako Before advice.

Implementace AOP je ve Spring možná pomocí proxy objektů, které "obalují" původní objekt a klientskému kódu se jeví, jako tento objekt.

Podpora AOP v Java EE 6 není tak rozsáhlá jako ve Spring. Definovat zde lze metoda (uvnitř jakékoliv třídy) nebo celý objekt, vykonávající kód interceptoru. Takový objekt nebo metoda mohou být typu `@AroundInvoke`, odpovídající around advice, nebo tří dalších speciálních typů. Těmi jsou vykonání operace po volání konstruktoru či před voláním destruktoru a interceptor pro časovače u EJB objektů. Použití interceptoru uvozuje v místě jeho vykonání anotace `@Interceptors(PouzityInterceptor.class)`.

Dalším způsobem je rozšíření jazyka Java s názvem AspectJ (používané Spring Roo), které lze uplatnit v prostém Java SE programu i v kombinaci s Java EE a Spring. AspectJ deklaruje aspekty jako třídy jazyka Java za pomoci anotací, nebo vlastním jazykem blízkým jazyku Java. Třída deklarovaná jako aspekt je buď automaticky detekována prostředím, nebo může být registrována v XML konfiguračním souboru [2]. Program využívající AspectJ je následně transformován na validní Java program.

V AspectJ příkladu 3.4 je aspekt typu Before advice, který před voláním metod určených výrazem `execution`, zavolá metodu `simpleBeforeAdvice()`. Pointcut `execution` v uvozovkách označuje volání všech metod (bez ohledu na návratový typ a argumenty) všech objektů v balíčku *com.test.service*.

MySimpleAspect.java

```
package com.test.aspect;

import org.aspectj.lang.annotation.*;

@Aspect
public class MySimpleAspect {
    @Before(value="(execution(* com.test.service.*.*(..)))")
    public void simpleBeforeAdvice(){
        System.out.println("This is a message from a before advice.");
    }
}
```

Obrázek 3.4: Příklad Before advice v Aspectj

3.1.2.1 AOP ve Spring Roo

Spring Roo hojně využívá mezi-typových deklarací (ITD), kdy aspekt (AspectJ objekt) doplňuje původní třídu o nové metody, proměnné, nebo rozhraní. Jeden objekt aplikace je rozšířen několika automaticky spravovanými objekty s různými rolemi.

V příkladu 3.5 je část mezi-typové deklarace vygenerované Spring Roo. Tento úryvek kódu rozšíří třídu `CartOrder` v souboru *CartOrder.java* o novou metodu `getInCart()` v souboru *CartOrder_Roo_JavaBean.aj*. Tato metoda se chová jako součást třídy `CartOrder`. Ve Spring Roo je tímto docíleno oddělení kódu spravovaného nástrojem a programátorem a také zpřehlednění kódu rozsáhlých objektů jeho rozdělením na menší logické celky.

CartOrder_Roo_JavaBean.aj

```
// WARNING: DO NOT EDIT THIS FILE. THIS FILE IS MANAGED BY SPRING ROO.

package com.rooshop.model;

import com.rooshop.model.CartOrder;

privileged aspect CartOrder_Roo_JavaBean {

    public Boolean CartOrder.getInCart() {
        return this.inCart;
    }
```

Obrázek 3.5: Příklad ITD

3.2 Model-View-Controller

Tento návrhový vzor byl původně vytvořen v sedmdesátých letech dvacátého století jako řešení problémů při manipulaci s rozsáhlými množinami dat v systémech tvořených v jazyce Smalltalk. Převážně je používán při tvorbě uživatelských rozhraní, lze však použít i u jiných systémů na bázi interakcí. Jeho účelem je vytvoření znovupoužitelného návrhu, oddělení vrstvy datové, prezentační a logiky zpracovávající uživatelem vyvolané události. Rozdělení problému interaktivní aplikace na samostatné podčásti vede k lepší flexibilitě, testovatelnosti a přehlednosti.

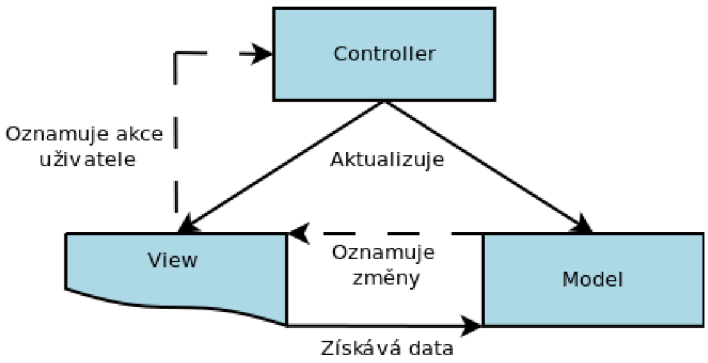
Na následujících řádcích bude představen MVC z pohledu internetových GUI aplikací, následně z pohledu kódu generovaného Roo a Forge.

- Model [7] definuje datovou strukturu aplikace. Obsahuje reprezentaci dat a business logiku, případně metody pro manipulaci s daty. Business logikou rozumíme logiku, kterou systém vykonává za účelem splnění pravidel a omezení požadovaných zákazníkem, například validační pravidla, nebo CRUD metody. Neměl by být přímo vázán na akce koncového uživatele. Neměl by také obsahovat kód sloužící pro prezentaci (například HTML). CRUD (create, read, update, delete) je akronymem pro základní operace proveditelné nad perzistentními daty, tedy vytvořit, přečíst, změnit a smazat. Používán bývá i v souvislosti s uživatelskými rozhraními umožňujícími tyto základní operace nad perzistentními daty vykonávat.

- View [7] prezentuje model ve formě požadované uživatelem. Obsahuje prezentační kód, v internetových aplikacích zpravidla (X)HTML kód společně s kódem skriptů. V prezentační části aplikace by se neměl nacházet kód přímo provádějící databázová volání, nebo kód zpracovávající požadavky

uživatele. View je zaměřen na zobrazení a formátování dat přijatých controllerem, nebo modelem a proto smí přistupovat k jejich proměnným a metodám.

- Controller [7] zpracovává přicházející požadavky od uživatele, proto smí přistupovat jak k proměnným a metodám modelu, tak k uživatelskému rozhraní, měnit jeho obsah. Může vytvářet instance objektů modelu a řídit jejich životní cyklus. Kód controlleru by neměl obsahovat databázová volání, ani prezentační kód.



Obrázek 3.6: Architektonický návrhový vzor MVC

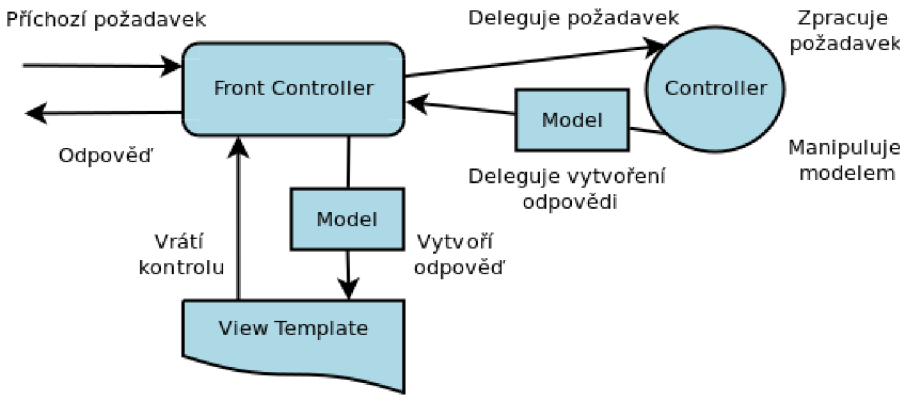
3.2.1 MVC a Spring ROO

Spring Roo ve svém základním nastavení umožňuje tvorbu MVC aplikace skrze framework Web MVC [2], který je součástí platformy Spring. Modulární přístup nástroje umožňuje volbu prezentační technologie změnit.

V tomto případě je model aplikace reprezentován POJO objekty využívajícími anotace Spring a JPA. Dále business a databázovou logikou vygenerovanou v souborech AspectJ. Framework umožňuje mezi modelem a controllerem vytvořit prostředníka, objekty servisní vrstvy, které zpřístupňují uživateli skrze svá rozhraní funkce definované případy užití.

Spring Web MVC implementuje vzor MVC ve formě zvané Front Controller. Roli Front Controlleru zde má DispatcherServlet. Jeho úkolem je směřování požadavků na další controllery aplikace, zobrazení výstupu uživateli, mapování témat vzhledu aplikace a jednotlivých stránek, nebo podpora nahrávání dat. Controller (objekt s anotací `@Controller` a mapováním požadavků na příslušné view) přijaté Http požadavky zpracuje a vyvolá metody servisní vrstvy. Pak vybere view, které bude použito pro zobrazení získaných dat a tato data mu poskytne skrze DispatcherServlet. Závislosti Controlleru na objekty modelu, jejichž proměnné a metody používá, jsou předány pomocí vkládání závislostí.

Prezentační část aplikace je v základním nastavení Spring Roo vytvořena pomocí JSPX, Java Server Pages kódu splňujícího specifikaci XML a Apache Tiles frameworku pro tvorbu šablon. Soubor JSPX nazýváme JSP dokument [20]. Apache Tiles umožňuje oddělit části kódu specifické pro jednotlivé stránky



Obrázek 3.7: Architektonický vzor Front Controller

od kódu společného. Tímto zefektivňuje vývoj View vrstvy. Soubor ve formátu JSP je převeden na servlet a vyplněn daty modelu. Zobrazení zajišťuje na aplikačním serveru rozhraní View nezávislé na použité technologii implementace zobrazení. Objekt View je pak vytisknut v požadované formě (XHTML, PDF, Excel) na výstup.

Technologie JSP umožňuje rychlou tvorbu dynamického webového obsahu nezávislého na platformě a použitém serveru [8]. Uvnitř souboru *.jsp můžeme nalézt například kód skriptletů, akcí, JSP EL (Expression Language) a (X)HTML, CSS.

Obsahem skriptletu [17] může být jakýkoliv kód v jazyce Java, od jednoduchých podmínek po business logiku aplikace. Toto svádí ke tvorbě nepřehledného kódu slučujícímu vrstvy MVC v jediném objektu. Nevýhodou je i problematická detekce chyb (špatně ošetřená výjimka skriptletu znamená zobrazení prázdné stránky bez chybové hlášky). Proto dnes použití skriptletů není doporučováno.

Akce [17] jsou, podobně jako skriptlety, vykonávány při načtení stránky. Jejich výhodou je snažší testovatelnost, znovupoužitelnost a odstraňování chyb.

Akce má tvar `<jsp:nazev_akce atribut="hodnota"/>`. JSP nabízí možnost, jak definovat vlastní akce. S využitím mechanismu rozšíření tagů pak tyto lze shromažďovat v knihovnách. Nejpoužívanější akce se staly standardem a byly shromážděny v knihovně JSP Standard Tag Library (JSTL).

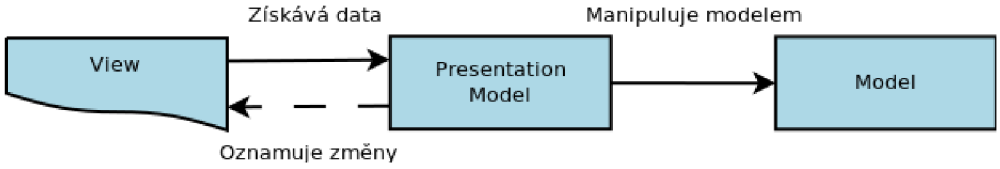
Od verze 2.0 JSP definuje, v porovnání se skriptlety, jednodušší způsob, jak přistupovat k objektům vně šablony. Je jím Expression Language (EL). Kombinace JSTL a JSP EL výrazů je dnes jedním ze standardních způsobů, jak vytvořit prezentační část Java EE aplikace. V závislosti, zdali chceme přiřadit, nebo číst hodnotu objektu ve výrazu, zapisujeme EL výraz `#{vyraz}`, nebo `${vyraz}` [8].

3.2.2 MVC a JBoss Forge

JBoss Forge je založeno na technoligích standardu Java EE 6. Aplikace v základním nastavení tvořena s pomocí MVC frameworku Javaserver Faces (JSF). Díky modularitě pak můžeme použít i jiné frameworky založené na JSF (např. RichFaces).

JSF uplatňuje "Presentation Model" [9], což je MVC vzor nazývaný oddělená prezentace (Separated Presentation). Jeho cílem je oddělit prezentační kód a kód modelu tak, aby prezentační vrstva mohla operovat s modelem a přitom model zůstal na ní nezávislý. Prezentační logika a stavy jsou spravovány speciálním objektem, který je řízen příslušným view. Tvorbu uživatelského rozhraní JSF skrze množinu

standardizovaných technologií, které nahrazují JSP. Jedná se o specifikaci komponent (například tlačítko, textové pole, tabulka), mechanismus pro jejich rozšíření, knihovny tagů pro manipulaci s těmito komponentami a podporu Expression Language. Framework uchovává stav uživatelského rozhraní, reaguje na události v něm vyvolané, překresluje jej a vyplňuje daty, také zajišťuje validaci a konverze dat [17].



Obrázek 3.8: Achitektonický vzor Presentation Model

Roli modelu v tomto případě zastávají POJO objekty využívající pro účely perzistence anotace JPA. Dále jsou JBoss Forge vygenerovány "backing bean" objekty reprezentující jednotlivá views (Presentation Model). Jedná se o JavaBeans komponenty, které obsahují data a funkce vázané na komponenty uživatelského rozhraní. Zajišťují CRUD funkcionalitu, správu konverzací (viz následující kapitola) a předávání dat pro zobrazení serverem.

Úlohu Front controlleru má FacesServlet [17], ten koordinuje tok požadavků mezi serverem a JSF aplikací.

View část generované aplikace je tvořena xhtml kódem doplněným o JavaServer Faces Expression Language prvky a speciální tagy JSF. Z pohledu JSF je view stromem UI objektů rozšiřujících třídu `javax.faces.component.UIComponent`. Tyto prvky jsou vázané na metody nacházející se v backing beans, nebo na hodnoty ve zdrojích dat.

3.3 Sezení a konverzace

Pojem sezení (Session) [21] je v informačních technologiích nejčastěji spojován se síťovou komunikací. Jedná se o časově omezenou výměnu informací mezi dvěma subjekty. V souvislosti s internetovými aplikacemi je důležitý pojem HTTP sezení, které uchovává své parametry v souborech cookie na straně klienta, nebo v URL. Těmito parametry jsou informace specifické pro dané sezení, jako nastavení vzhledu, nakupované položky, určení uživatele, identifikace objektu, který sezení reprezentuje na straně serveru. Je tak možné uchovat kontext komunikace. Sezení je podporováno ve specifikacích všech moderních webových frameworků a jazyků. Jeho konec nastává zpravidla po vypršení určitého časového limitu od poslední aktivity klienta.

Evolucí sezení jsou konverzace. Nabízí řešení v případech, kdy je rozsah jediného požadavku příliš malý a rozsah celého sezení naopak příliš velký. Jejich účelem je obsloužit specifické série požadavků (toky, nebo také proudy) [21] [19] tvořící logický celek (osazení nákupního košíku -> vyplnění formuláře objednávky -> potvrzení objednávky). Lze s nimi přesně definovat způsoby, jakými se uživatel může dostat ke každému z kroků prováděné činnosti. Kromě řízení navigace je další výhodou například uchovávání informací o otevřených lištách uživatele a možnost vést několik paralelních konverzací. Tyto vlastnosti

vedou k snadnější implementaci případů užití, kdy jeden uživatel potřebuje vykonávat více souběžných akcí.

Podporu konverzací je možné nahradit přenosem informací v sezeních. Nevýhodou zde je, že data s platností jen několik požadavků, uchovávaná v sezeních, není snadné spravovat. Tyto pak mohou zůstat nesmazány až do vypršení sezení. Na službě provozované spojením více serverů je často potřeba distribuovat probíhající sezení. Pokud sezení obsahuje velké množsví dat, je celá služba nadměrně vytěžována. Konverzace toto množství dat snižuje díky své kratší životnosti. Délka konverzace může být jeden požadadavek až celá doba sezení.

3.3.1 Sezení a konverzace v Java EE 6

Konverzace jsou součástí specifikace CDI a JSF. Beans, které konverzace používají, je mají vložené anotací `@ConversationScoped` [18]. Takový objekt má životnost jedné konverzace. Průběh konverzace je implementován jako posloupnost metod bean objektu volaných šablonou. Metoda v reakci například na odeslání formuláře vykoná požadované operace, jako validace a úprava modelu, nebo přesun na následující stránku. Další typy CDI anotací definující rozsah platnosti (scope) objektů při interakci jsou `@SessionScoped`, `@ApplicationScoped`, `@RequestScoped`, `@Dependent` [18]. `@SessionScoped` objekt má životnost po dobu sezení mezi klientem a serverem. Ostatní zmíněné anotace podle názvu přiřazují objektu životnost po dobu běhu aplikace (`@ApplicationScoped`), jediného požadavku (`@RequestScoped`), po dobu života objektu, který jej vytvořil (`@Dependent`). Framework JSF oproti CDI přidává `@ViewScoped` rozsah s životností počínající prvním požadavkem na stránce, která používá `@ViewScoped` objekt, a končí s první akcí provedenou na stránce následující. Uplatnění má v komplexnějších formulářích, asynchronně komunikujících se serverem.

3.3.2 Sezení a konverzace ve Spring

Podporu pro konverzace Spring nabízí ve frameworku Spring Web Flow [19], kterým je možné rozšířit zavedené frameworky Javy EE (Struts, JSF, Spring MVC a jiné). Standardně Spring definuje 5 druhů rozsahu platnosti bean objektu (Singleton, Prototype, Request, Session, Global Session) [19], které Web Flow rozšiřuje o konverzace. Objekt Singleton má pouze jedinou instanci v rámci kontextu Spring. Objektu Prototype je vytvořena nová instance s každým jeho vložením, ale nepodléhá automatické správě paměti. Vlastnosti Request a Session jsou srovnatelné s těmi definovanými Java EE 6. Global Session souvisí s technologií portletů, která není předmětem této práce. Rozsah platnosti je definován buď anotací, například `@Scope("prototype")`, nebo v konfiguračním souboru *ApplicationContext.xml*.

Základním prvkem Spring Web Flow je tok (flow) [21], definovaný v XML souboru *nazev-proudu.xml* ve složce *WEB-INF/flows*. Obsah toku je uchováván ve snímcích (snapshots). Toto usnadňuje vracení se k předchozím úkonům jednoduše vyvoláním předchozího snímku.

Tok je implementován pomocí elementů stavů (state), akcí (action) a přechodů (transition) mezi stavy. Stavy popisuje konfigurační soubor toku. Akcí je například zobrazení stránky, nebo vykonání metody controlleru pomocí Spring Expression Language. Uvnitř stavu se nachází mimo jiné element přechodu `transition`, řídící navigaci mezi stránkami tvořícími tok.

Rozlišujeme několik druhů stavů, v souboru uvozených stejnojmennými elementy: `start-state`,

`end-state`, `view-state`, `action-state`, `decision-state` [19]. Nejvýznamnějším je `view-state`, který má na starost modelování stránky. Po jejím zobrazení je výkon proudu pozastaven, dokud nevyvstane nová událost v uživatelském rozhraní. Stavy action a decision obalují akci a přechody, nejsou vázány na zobrazované stránky. Rozdílem mezi nimi je použití stavu decision výhradně pro vyhodnocení podmínky (if-then-else). Stavy end a start jsou použity pro operace vykonávané na začátku a konci proudu (například počáteční inicializace objektů).

Spring Web Flow rozšiřuje možnosti rozsahu platnosti objektů o request, flash, view, flow, conversation, spravované frameworkem. Rozsah platnosti je přiřazován v konfiguračním souboru toku používaným objektům v atributech.

Request má platnost jednoho požadavku uvnitř proudu. Flash rozšiřuje platnost do doby zobrazení svého obsahu na výstup. Má použití například pro varování a chybové hlášky. View rozsah atributu je omezen začátkem a koncem view stavu, uvnitř kterého je umístěn. Flow je platný po dobu celého proudu, avšak nepřístupný z dalších proudů uvnitř proudu vytvořených. Conversation atribut má obdobně rozsah celého proudu, ale je v rámci proudu platný globálně.

3.4 Active Record

Tento pojem se poprvé objevil v knize „Patterns of Enterprise Application Architecture" autora Martina Fowlera [10]. Jedná se o architektonický návrhový vzor využívaný pro účely perzistence a objektově-relačního mapování (ORM). Instance objektu Active Record reprezentuje řádek databázové tabulky. Obsahuje logiku přímo komunikující s databázovou technologií (CRUD, vyhledávání v databázi), v praxi většinou bez nutnosti znát databázový jazyk. Jakákoliv akce nad instancí je pak automaticky provedena nad příslušným řádkem databáze. Uplatnění tento vzor nachází v technologiích zaměřených na rychlý vývoj méně komplexních aplikací, kde kód business logiky není nutné oddělit od kódu entit (Ruby on Rails, Zend Framework).

3.4.1 Active Record ve Spring Roo a JBoss Forge

Ve svém základním nastavení využívá Spring Roo vzor JPA Active Record. Entity modelu aplikace jsou rozšířeny o aspekty zaměřené na perzistenci (jedná se o AspectJ ITD soubory), umožňující základní databázové operace [3].

Aplikace vytvořená v JBoss Forge Active Record vzor nevyužívá, operace s databází jsou obsaženy v backing beans.

3.5 Convention over Configuration a Don't repeat yourself

CoC je softwarové paradigma propagující zjednodušování bez omezení flexibility. Na jeho základě by měl programátor definovat pouze nekonvenční aspekty aplikace, nástroje pak podle zavedených konvencí doplní zbylé části. Pokud aplikované konvence nevyhoví požadavkům, lze je explicitně změnit [2].

Podobně DRY je princip snažící se omezit duplicitu úkonů a informací při vývoji software. Říká, že uvnitř vytvářeného systému by měl být pouze jeden unikátní představitel pro každý druh informace

[2]. Příkladem je AOP, kdy aspekty obalují informace používané na více místech systému, nebo inverze kontroly, kdy lze snadno změnit měnit implementaci části projektu bez výrazných zásahů do zbytku kódu.

Oba přístupy se snaží vést k čitelnějšímu kódu, snažšímu vývoji a omezení boilerplate kódu, jenž je vzhledem k složitosti své funkcionality obsáhlý a nelze zapsat jednodušeji.

3.5.1 CoC a DRY ve Spring Roo a JBoss Forge

Spring Roo a JBoss Forge toto uplatňují například v rámci scaffoldingu, kdy uživatel definuje použité technologie a doménový model, nástroje podle zavedených konvencí vytvoří aplikaci. Podobně nástroj Apache Maven předpokládá konvenční strukturu projektu, parametry kompilace, parametry tvorby balíčků atd. Tato nastavení je pak možné změnit editací souboru pom.xml, nebo příslušnými parametry v příkazovém řádku.

3.6 Scaffolding a metaprogramování

Pojem metaprogramování [13] lze definovat jako tvorbu programů, které vytvářejí programový kód, nebo jím manipulují. Jeho výhodami jsou lepší udržovatelnost kódu a zvýšená rychlost vývoje. Mezi programy vytvářející kód patří například preprocesory, kompilátory, generátory parserů a také Spring Roo a JBoss Forge. Společnou vlastností těchto programů je vstup na vyšší úrovni abstrakce, než vytvořený kód.

Scaffolding je metaprogramovací technika používaná pro zvýšení produktivity v počátečních fázích vývoje software. V kontextu této práce umožňuje na základě definice automaticky vytvořit MVC aplikaci se základní funkcionalitou. Definicí může být stávající databáze, nebo množina příkazů popisujících model aplikace a konfigurace. Výsledkem je pak aplikace obsahující základní funkce. Popularitu získala díky Ruby on Rails, odkud se rozšířila do dalších frameworků.

Kapitola 4

Srovnání s technologií založenou na jiném jazyce

Jak bylo zmíněno v kapitole 2.2, existuje dnes množství frameworků a nástrojů podporujících rychlý vývoj internetových aplikací. Většina jich je založena na skriptovacích jazycích. Pro referenční účely bude v této kapitole nejdříve představena technologie založená na jazyce Python, RAD framework Django [14], poté budou uvedeny hlavní rozdíly mezi porovnávanými RAD nástroji založenými na jazyce Java (společně s technologiemi JSF, Spring MVC, Hibernate) a Djangem.

4.1 Python a Django

Jazyk Python je objektově orienovaný interpretovaný jazyk, dynamicky typovaný s automatickou správou paměti. Jeho první specifikace vyšla v roce 1994, dnes je již ve verzi 3.3.0. Python je zaměřený na snadnou čitelnost kódu, jednoduchost a explicitnost.

Django je webový MVC framework založený na jazyce Python, snažící se automatizovat maximum úkonů vykonávaných při vývoji aplikací. Projekty vytvořené Djangem jsou složené ze snadno přenositelných modulů, aplikací nižší úrovně. Moduly mohou obsahovat podporu různých aspektů, jako autentifikace uživatele, sezení (sessions), modulem je i samotná MVC aplikace. Další z důležitých vlastností frameworku jsou automatická validace uživatelského vstupu a vývojový server. Django byl uvolněn pro veřejnost tvůrci v roce 2005. Jeho poslední verze 1.4.x podporuje jazyk Python ve verzi 2.5.x až 2,7.x. Vytvořené aplikace jsou nasaditelné na HTTP serverech.

4.2 Ovládání, tvorba aplikace

Po instalaci frameworku Django je projekt vytvořen v příkazové řádce spuštěním administrační utility *django-admin.py* s parametrem `startproject nazev_projektu`. Tento příkaz vygeneruje soubory představující základní konfiguraci a správu projektu. Další ovládání frameworku probíhá pomocí vygenerovaného skriptu *manage.py*, který zprostředkovává administraci v rámci projektu. Kostra MVC aplikace je vytvořena příkazem `startapp`. Doplnění kódu prvků model, view, controller do příslušných připravených souborů je v rukou vývojáře. Systém objektově-relačního mapování vytvoří po zadání

	Django	JBoss Forge	Spring Roo
Jazyk	Python	Java	Java
Šablony	Ano, vlastní systém.	V základu JSF.	V základu Apache Tiles - JSP.
Usnadnění konfigurace	Pouze vygeneruje obecné konfigurační soubory.	Oba nástroje vytvoří kompletní konfiguraci podle zadaných parametrů.	
Generování kódu	Pouze připraví prázdné soubory.	Oba nástroje podle zadaných parametrů vytvoří prototyp připravený k nasazení.	

Tabulka 4.1: Srovnání Django, JBoss Forge, Spring Roo

příkazu `syncdb` databázové tabulky pro data použitých modulů. S obsahem modelu aplikace pak lze manipulovat v rozhraní interpretu jazyka Python, nebo v oknu prohlížeče, pokud použijeme modul pro administraci projektu.

4.3 MVC a architektura frameworku

Django nazývá svou MVC architekturu Model-View-Template, kde View (v Django) odpovídá Controlleru (v MVC architektuře) a Template (v Django) odpovídá View (v MVC arch.). Model obsahuje reprezentaci dat a business logiku. Úkolem View části aplikace je zpracování požadavků a koordinace dat získávaných z modelu, prezentační logika. Na základě požadavku předá data z modelu pro vykreslení šabloně. Systém šablon kombinuje jazyk HTML a šablonovací jazyk frameworku Django.

Zpracování požadavků ve frameworku má následnou podobu. Web server posílá požadavky od uživatele aplikaci a směrem opačným. V aplikaci má jejich směřování za úkol Django Middleware. Jedná se o systém nízkoúrovňových pluginů pro globální úpravu vstupů a výstupů frameworku Django [15]. Příchozí požadavky jsou odeslány jednotlivým view objektům na základě mapování určeného v souboru *urls.py*, nezávisle na jejich implementaci. View s použitím modelu a šablony vytvoří odpověď. Tato odpověď je zpracována příslušnou komponentou middleware, poté odeslána serveru. Operace nad databází provádí Django ORM skrze Python Database API a jeho rozhraní implementující adaptér přístupu k databázi.

4.4 Srovnání

Narozdíl od modulárních technologií Javy EE je Framework Django monolitický, ne však striktně. Spojuje šablonovací systém pro prezentaci, objektově-relační mapování, autentifikaci a další technologie do jednoho celku. Lze použít například alternativní systém šablon.

Následují rozdíly ve tvorbě aplikací mezi Spring Roo, JBoss Forge a Djangem. Administrační nástroj frameworku Django neposkytuje tak široké možnosti uplatnění během psaní kódu aplikace. Generování kódu s jeho použitím probíhá jednorázově na samotném začátku projektu. Dále je tvorba kódu pouze v rukou programátora. Administrační nástroj pak slouží zejména pro operace s databází a jiné podpůrné kroky při vývoji aplikace. Se Spring Roo a JBoss Forge může programátor upravovat kód tvořené aplikace během vývoje. Vzhledem k nižšímu počtu znaků potřebnému pro popis stejného chování programu v jazyce Python v porovnání s jazykem Java, není nemožnost automatického generování kódu aplikace ve

frameworku Django výraznou nevýhodou. Obzvláště na úrovni frameworků, jako Spring Web MVC, JSF a Django. Všechny tři technologie využívají šablon ke tvorbě dynamického obsahu a oddělení prezentační části od modelu.

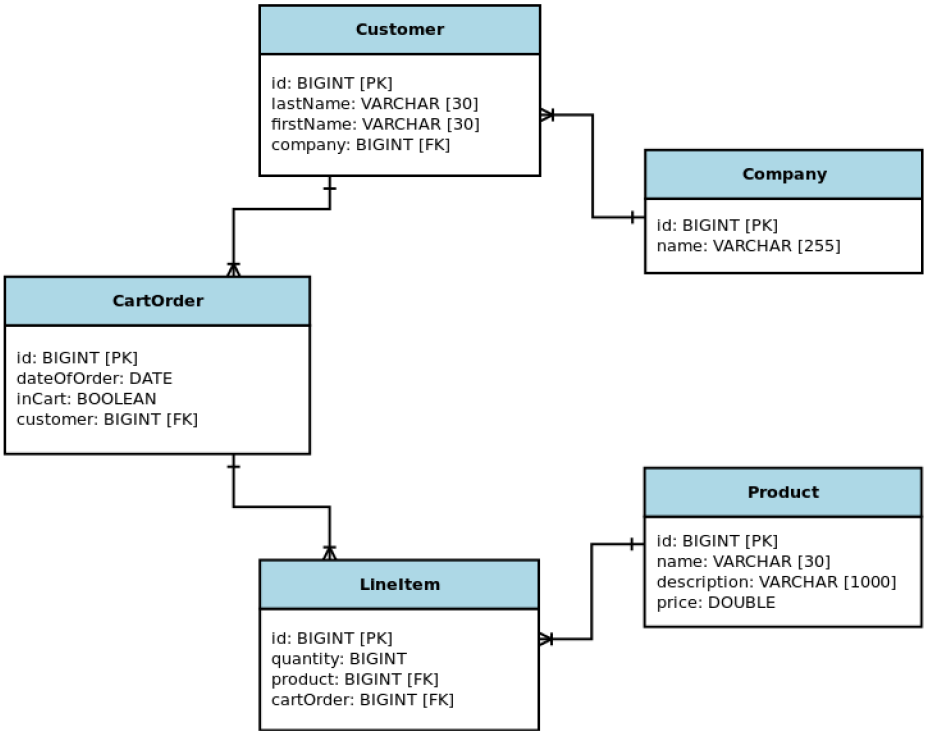
Django řadí mezi své filozofie [14] principy DRY a Explicit is better than implicit (explicitně je lépe než implicitně). Princip DRY je uplatněný jak při vyvoji frameworku, tak podporován při jeho používání. Paradigma Convention over Configuration zde není ve velké míře uplatňováno. V Djangu je potřeba i základní konfiguraci provést manuálně.

Kapitola 5

Vytvoření vzorové aplikace

Tato kapitola se zaměřuje na praktické porovnání Spring Roo a JBoss Forge.Na následujících stranách bude představeno jejich ovládání při tvorbě dvou jednoduchých informačního systému se stejnou funkcionalitou.

Pro lepší orientaci v textu této kapitoly je v příloze A a B posloupnost příkazů vedoucích k vytvoření vzorových aplikací a v příloze C a D adresářová struktura vygenerovaného projektu. Pro detailnější náhled do struktury a zdrojového kódu je k dispozici repozitář (http://sourceforge.net/p/roovsforgecmp/wiki/Home/) a přiložené CD. Kvůli účelům srovnání byly aplikace ponechány ve stavu, v jakém je vygenerují Spring Roo a JBoss Forge. Výjimkou je úprava konfigurace nutná pro nasazení aplikace vygenerované Spring Roo (viz kap. 5.5).
Vzorem projektu je jednoduchý příklad obchodu představený v knize Getting Started with Roo [21] viz obrázek 5.1.



Obrázek 5.1: Diagram databáze vzorové aplikace

Zákazníci (customer) jsou zaměstnáni společností (company). Každý zákazník může do svých objednávek (cart_order) vkládat položky (line_item), které zapouzdřují samotné produkty (product) na prodej. Příklad testuje podporu základních datových typů a nejčastěji používané relace 1:M.

Porovnávány byly JBoss Forge 1.2.1 a Spring Roo 1.2.3. Jako operační systém byl použit Ubuntu linux 12.04, jako aplikační server JBoss AS 7.1.1 a verze javy Java Oracle JDK 7.

Pro uchování dat byla použita databáze H2, která je součástí JBoss AS 7. Připojení k databázi zprostředkoval server skrze Java Naming and Directory Interface (JNDI) referenci ExampleDS, předdefinovanou v konfiguračním souboru na serveru. Součástí její konfigurace jsou informace potřebné k připojení k databázi, například přihlašovací údaje, použitý typ a ovladač databáze, umístění této reference. V nastavení perzistence aplikace pak stačí deklarovat použití této reference a cestu k ní. Konfigurace připojení k databázi vně balíčku aplikace zlepšuje její přenositelnost mezi různými prostředími. ExampleDS není díky použité databázi doporučován pro nasazení v produkčním prostředí, ale plně postačuje pro tuto práci.

Pro účely ORM aplikaci posloužil framework Hibernate, implementující specifikaci Java Persistence API 2.0. Hibernate patří mezi nejrozšířenější technologie pro objektově-relační mapování, je vyvíjen společností Red Hat.

Pro prezentaci a manipulaci s modelem aplikace byly použity Spring MVC (ve Spring Roo) a JSF (v JBoss Forge). Jména projektů jsou rooshop a forgeshop.

5.1 Instalace a spuštění nástroje

Oba nástroje jsou aplikacemi jazyka Java, distribuované v archivech ZIP. Prekvizitou použití je nainstalování Java Development kit verze 6 nebo vyšší a systémová proměnná $JAVA_HOME odkazující na jeho domovský odresář. Instalace na operačním systému Linux (či unix) spočívá v rozbalení archivu a vytvoření symbolického odkazu na spustitelný soubor aplikace v systémové složce */usr/bin*. Ve Windows je potřeba vytvořit proměnnou prostředí odkazující na adresář se spustitelným souborem. Po vykonání těchto úkonů je nástroj spuštěn příkazem v příkazové řádce.

5.1.1 Specifika Spring Roo

Nedává uživateli možnost ze svého rozhraní navigovat a provádět úkony v adresářové struktuře, proto musí být práce s ním započata v předem vytvořené domovské složce budoucího projektu.

5.1.2 Specifika JBoss Forge

Nabízí základní příkazy unixového shellu pro navigaci a operace v adresářové struktuře, jako například cd, grep, ls, mkdir, pwd, rm, touch. Není proto nutné prostředí nástroje opouštět při provádění jednoduchých systémových operací.

5.2 Vytvoření a nastavení projektu, systém nápovědy

Vytvoření projektu a konfigurace perzistence je s porovnávanými RAD nástroji obsaženo ve dvojici příkazů (viz přílohy A, B).

Urychlit tvorbu projektu a vybrat správné nastavení pomáhá systém nápovědy a rychlého doplňování textu. Oba nástroje v rámci nápovědy podporují zobrazení aktuálně použitelných příkazů a parametrů stiskem TAB. Tato klávesa slouží i k rychlému automatickému dokončování rozepsaných příkazů a jejich parametrů. Jednoduchý příklad: `ent` + TAB doplní klíčové slovo `entity`, další stisk klávesy doplní v případě JBoss Forge parametr `--named`, uživatel pokračuje doplněním názvu. Dále je k dispozici použití `help příkaz`, které o tomto příkazu zobrazí dostupné informace.

5.2.1 Specifika Spring Roo

Ve Spring Roo slouží pro inicializaci nového projektu příkaz `project` s název balíčku tvořené aplikace. Tento příkaz dá vzniknout souboru *pom.xml* (viz kapitola 2.4) v kořenovém adresáři projektu a *ApplicationContext.xml* se základním nastavením Spring ve složce **src/main/resources/META-INF/Spring*. Následuje deklarace perzistence (`jpa setup`). Zvolena je databázová technologie a poskytovatel perzistence v aplikaci. Po tomto příkazu Spring Roo aktualizuje stávající soubory a vytvoří *persistence.xml (v *src/main/resources/META-INF/)* s nastavením poskytovatele.

Co se nápovědy týká, pouze Spring Roo nabízí kdykoliv za běhu nápovědu (příkaz `--hint`), důsledně popisující další možné kroky při tvorbě aplikace.

5.2.2 Specifika JBoss Forge

V JBoss Forge podobného výsledku docílíme příkazem `new-project`, po jehož zavolání bude vytvořena základní adresářová struktura a soubor *pom.xml*. Perzistence se nastaví příkazem `persistence setup` s parametry poskytovatele perzistence a aplikačního kontejneru. Podporovány momentálně jsou aplikační servery JBoss AS a Oracle Glassfish. V jiných případech se namísto kontejneru specifikuje pouze typ rozhraní komunikace s databází (JDBC, JTA, JNDI Datasource objekt), dalšími parametry dodatečné údaje jako např. uživatel, heslo, jdbc driver a URL pro JDBC spojení a typ databáze. S jejich potvrzením JBoss Forge aktualizuje *pom.xml* a vygeneruje *persistence.xml* se zvoleným nastavením. Pokud budou v projektu aplikována JPA validace, nastaví ji `validation setup`.

5.3 Deklarace modelu aplikace a jeho editace

Tvorba modelu aplikace je v obou nástrojích podobná. Nejprve je vytvořena entita (příkaz `entity`), v ní dalšími příkazy jednotlivé atributy a relace včetně JPA validačních omezení. Dále je podporován jednoduchý přesun mezi entitami při jejich tvorbě a editaci.

5.3.1 Specifika Spring Roo

Umístění entity v rámci balíčku a její jméno je určeno parametrem `--class` příkazu `entity`. Následuje deklarace proměnných uvnitř entity (příkaz `field`). Těmto je přiřazeno jméno, typ a JPA omezení.

Speciálním případem je deklarace relací. Tu je nejlépe představit na příkladu, například u entity Customer (příloha A, řádek 10).

Parametr set značí datový typ pole `Set`, `--type` určuje datový typ objektů, `--cardinality` určuje typ relace, reprezentovaný ve formě příslušné JPA anotace. Parametr (podle stejnojmenného parametru

u JPA anotace `@OneToMany`) určuje vlastnictví relace. CartOrder je vlastníkem obousměrné databázové vazby, nese cizí klíč `customer`. Poté je potřeba změnit aktuální editovanou třídu příkazem `focus` a dokončit relaci na druhé straně. U entity `CartOrder` zbývá doplnit cizí klíč (viz příloha A, řádek 12).

Příkaz vytvoření entity dá vzniknout pěti souborům. Prvním souborem je POJO *NazevEntity.java*, tedy definice entity, její proměnné, anotace JPA a anotace Spring Roo. Dalšími jsou AspectJ ITD soubory doplňující tento objekt. Jedná se o soubor Active Record, soubor s nastavením identifikace entity a samostatné soubory rozšiřující objekt o šablonu metody `toString()` a anotaci `@Configurable`. S deklarací atributů entity je vygenerován dále soubor *NazevEntity_Roo_JavaBean.aj*, nesoucí metody typu `get()` a `set()`.

Pro editaci již vytvořených entit je potřeba použít textový editor nebo vývojové prostředí. Spring Roo dynamicky reaguje na provedené změny. Po smazání proměnné jsou automaticky smazány i odpovídající metody, které s ní pracují. Uživatel pak redefinuje proměnnou v příkazové řádce nástroje, nebo v editoru, a metody jsou opět vygenerovány. Odpovídajícím způsobem nástroj automaticky edituje i prezentační kód.

Pokud je smazána celá entita, jsou spolu s ní automaticky smazány podpůrné AspectJ soubory společně s vygenerovaným kontrolérem. Pokud není v plánu její opětovné vytvoření, pozůstatky musí být tvůrcem odstraněny z objektů, které s ní byly v relaci. Musí být opraven, nebo znovu vygenerován i případný kód prezentační (viz kap. 5.4.1).

5.3.2 Specifika JBoss Forge

I v tomto případě je použito příkazu entity pro vytvoření entity, způsob zadání jména a jejího umístění je u obou nástrojů velmi blízký. Podobným způsobem jako u Spring Roo probíhá i tvorba proměnných modelu. Pro srovnání příklad tvorby shodné relace v JBoss Forge (relace mezi `CartOrder` a `Customer`) viz příloha B, řádek 16.

Hlavním rozdílem mezi nástroji je automatické vytvoření inverzní strany relace v JBoss Forge. Do objektu `CartOrder` je zapsána proměnná `customer`. Tvorba entity a jejích proměnných dá vzniknout jedinému souboru (např. *Customer.java*). Jedná se o Java Bean objekt s funkcemi typu `get()` a `set()`, `toString()` a JPA anotacemi. JPA omezení jsou přidávána příkazem `constraint`, kdykoliv při modelování entity.

Editace entity a jejích proměnných v Jboss Forge je vykonávána z příkazové řádky způsobem známým uživatelům shellu. Celé entity, či jejich proměnné, maže příkaz `rm`. Přehled proměnných a metod zobrazí příkaz `cd`. Jboss Forge nemá podporu automatického monitorování úprav provedených z vně nástroje, ani podporu automatické editace souborů projektu. Při smazání atributu (Jboss Forge verze 1.1.3 a nižší) nedochází k jeho úplnému odstranění. Zbytky kódu zůstanou v metodě `toString()`. Po odstranění celého objektu entity nejsou upraveny žádné z ostatních souborů. Pokud byl objekt v relaci s jinými objekty, je nutné provést ruční zásah do kódu entit s ním v relaci. V obou případech je potřeba manuálně upravit, nebo opět vytvořit prezentační kód (kapitola 5.4.2).

5.4 Nastavení MVC Frameworku, scaffolding a tvorba balíčku

Posledními kroky při tvorbě aplikace jsou vygenerování uživatelského rozhraní na základě definovaného modelu, kompilace a její sestavení do balíčku.

5.4.1 Specifika Spring Roo

Příkaz `web nazev_frameworku setup` provede nastavení, vygeneruje šablony a grafiku podle pravidel struktury projektu Maven a archivu WAR ve složce **/src/main/webapp/*. Tentýž příkaz s parametry `all --package` vytvoří ke všem entitám modelu odpovídající stránky ve složce **/src/main/webapp/* a kontroléry na místě určeném parametrem `--package`. Posledním příkazem je `perform package`, který spustí nástroj Apache Maven, provede kompilaci a sestaví balíček s aplikací do adresáře **/target*.

5.4.2 Specifika JBoss Forge

Příkaz `scaffold` s parametrem `setup` a `indexes` vykoná tytéž úkony, jako `setup` ve Spring Roo. Generování uživatelského rozhraní pak dokončí `from-entity` tvorbou šablon specifických pro entity projektu. Kompilaci a tvorbu balíčku v adresáři **/target* vykoná příkaz `mvn package`.

5.5 Úprava aplikace ve Spring Roo pro JBoss AS 7

Aplikace vytvořená s pomocí Spring Roo je připravená pro nasazení na webovém kontejneru. Nastavena je, aby jí používané moduly Java EE běžely na straně aplikace. Aplikační server JBoss AS tyto moduly provozuje na své straně. Proto je nutné přizpůsobit konfiguraci jednomu z těchto přístupů. Pro práci bylo zvoleno nastavení, které upřednostňuje služby JBoss AS, nevyžaduje dodatečnou konfiguraci serveru a je použitelné na jeho základním profilu Java Enterprise Edition 6 web profile.

Vytvořená aplikace má nastavenou jako jednotku perzistence `LocalContainerEntity ManagerFactoryBean` frameworku spring. Konfigurace v souboru *persistence.xml* způsobí ale také vytvoření druhé jednotky serverem. Toto vede k chybě, znemožňuje nasazení. Následující řešení zabrání vzniku jedné z jednotek.

1. Zaregistrování jednotky perzistence v konfiguračním souboru *web.xml*.
2. Odstranění jednotky perzistence Spring z *ApplicationContext.xml* a nastavení JNDI vyhledávání pro jednotku novou.
3. Doplnění JNDI zdroje dat do *persistence.xml*. Zdrojem dat pro tento projekt je reference `ExampleDS`.

5.6 Vygenerované uživatelské rozhraní

Vytvořená uživatelská rozhraní (viz přílohy G až J) poskytují CRUD funkcionalitu, neobsahují výrazné chyby. V obou rozhraních ale můžeme nalézt vlastnosti, které před komerčním nasazením aplikace potřebují upravit.

Pokud v JBoss Forge deklarujeme validační omezení, vyskytne se v uživatelském rozhraní chyba, kdy při vytváření nové položky v aplikaci je povinné vyplnit i vlastnosti jiného objektu, který je s tímto v relaci (platí pro omezení omezení NotNull). Vytváření jedné nové entity znamená vyplnit i proměnné entity v závislosti (viz příloha K). Objekt v relaci, kterému byly vyplněny proměnné, pak není vytvořen. V šablonách je také drobná chyba, kdy je na základě JPA validačních omezení nastaveno omezení textového pole. Atribut tohoto omezení obsahuje syntaktickou chybu. Jedná se o atribut `maxlength` u tagu `<h:inputText>`, viz příloha M.

Nepříjemností uživatelského rozhraní vygenerovaného Spring Roo je zobrazování relací pouze ze strany vlastníka. Podobně jako v předchozím případě, i zde je finální způsob zobrazení relace mezi dvěma entitami nechán na vývojáři.

Další vlastností obou aplikací, kterou je vhodné pozměnit, jsou metody `toString()`. Vhodná textová reprezentace objektu není triviální záležitostí a nástroje neznají způsob, jak ji správně odvodit, nebo nechat deklarovat. Prioritu mají při tvorbě metody `toString()` číselné datové typy, proto v praxi nástroj může zvolit pro identifikaci objektu například počet položek v košíku, nebo cenu. Textová reprezentace objektu ve Spring Roo je součástí AspectJ souboru, Spring MVC kontroléru. Tvůrci nástroje nedoporučují tyto soubory upravovat. Doporučováno je takové úpravy provést po odstranění mezitypových deklarací, buď manuálním, nebo funkcí push-in vývojového prostředí Eclipse (kap. 5.1.3).

5.7 Shrnutí

Ovládání obou nástrojů je velmi podobné s drobnými odlišnostmi. JBoss Forge poskytuje širší možnosti nastavení validace a dovoluje ze svého rozhraní napsat vlastní zprávy o validačních omezeních. Tyto zprávy v nasazené aplikaci nahrazují původní automaticky vytvořené zprávy. Výhodou Spring Roo je automatická detekce změn a úprava souborů na základě těchto změn. Tento systém je ve Spring Roo nastaven konzervativně. Pokud jednoznačně nelze určit, jaké úpravy mají být automaticky provedeny, nejsou provedeny žádné.

Při tvorbě modelu se oba nástroje ukázaly rovnocenné a velmi efektivní. 1818 znaků kódu JBoss Forge vytvořilo 13828 znaků kódu modelu v pěti Java souborech reprezentujících pět entit. 1754 znaků kódu Spring Roo vytvořilo 28749 znaků v 30 souborech Java a AspectJ popisujících tentýž model. Na editaci atributu uvnitř entity Spring Roo reaguje podle specifikace a nenechává nežádoucí kód. Smazání atributu v JBoss Forge znamená drobné manuální úpravy kódu. Úpravy kódu aplikace po smazání celé entity nechávají oba nástroje na vývojáři.

Uživatelské rozhraní vytvořené nástroji je funkční. Má úlohu prototypu a před konečným nasazením, obzvlášť komerčním, vyžaduje větší množství zásahů do prezentačního kódu, kódu kontrolerů a prezentačního modelu.

Kapitola 6

Podpora ve vývojových prostředích

Nástroje JBoss Forge a Spring Roo jsou dostupné jako samostatné nástroje, lze je tedy používat v kombinaci s jakýmkoliv editorem, nebo vývojovým prostředím. Začlenění do vývojového prostředí podporujícího technologie Java EE 6 a Spring zvyšuje komfort a efektivitu práce. Výhodami prostředí jsou například automatické doplňování kódu, syntaktická kontrola, správa verzí, sjednocení s dalšími nástroji, jako Apache Maven, aplikační server. Porovnána bude podpora hlavními vývojovými prostředími, Eclipse (Eclipse Java EE IDE for Web Developers, Juno Service Release 2) a také jeho verze pro vývoj ve Spring Framework, Spring Tool Suite (verze 3.2.0), dále IntelliJ Idea (verze 12.0.4), Netbeans (verze 7.3).

Bylo zkoušeno vytvoření nového projektu (pokud prostředí integruje Spring Roo nebo JBoss Forge), import již existujícího projektu, sestavení nástrojem Maven, nasazení na integrovaný aplikační server. V této kapitole je také popsáno odstranění mezitypových deklarací specifických pro Spring Roo v prostředí Eclipse. JBoss Forge po své činnosti nezanechává žádný atypický kód.

6.1 Popora vývoje se Spring Roo v Eclipse IDE

Prostředí Eclipse je v současné době nejvíce optimalizováno pro vývoj s pomocí Spring Roo. Společnost SpringSource vyvíjí vlastní předkonfigurovanou verzi Eclipse, Spring Tool Suite (STS), rozšířené o plnou podporu jejich technologií. Nic nebrání k instalaci těchto rozšíření v odpovídající verzi prostředí Eclipse.

6.1.1 Projekt Spring Roo v základní instalaci Eclipse

Pokud vývojář nechce, nebo nemůže používat STS, existuje několik možností s různým stupněm integrace nástrojů. Před importem existujícího Spring Roo projektu do základní instalace prostředí Eclipse (neobsahující podporu pro RAD nástroje, Apache Maven, JBoss AS) je nutné zavolat ve Spring Roo příkaz `perform eclipse`, který vygeneruje a nakonfiguruje artefakty vývojového prostředí. Aplikace bude Java EE webovým projektem.

Podporu Apache Maven přidá Maven Integration plugin (http://eclipse.org/m2e/). Rozšíří základní instalaci o podporu jeho projektů, mimojiné umožňuje pokročilou editaci *pom.xml* a ovládání nástroje pro tvorbu buildů z vývojového prostředí. Dále je nutné nainstalovat AJDT plugin pro podporu AspectJ (http://www.eclipse.org/ajdt/) a z téhož zdroje plugin pro konfiguraci Apache Maven projektů. Pokud

projekt není nakonfigurován pro AspectJ, není Maven Integration plugin schopen rozpoznat nastavení a projekt sestavit.

Podporu JBoss AS a další produkty firmy Red Hat integruje JBoss Tools plugin. Po registraci v záložce Servers lze server ovládat z vývojového prostředí a aplikaci nasadit na jeho zaregistovanou instanci. Jsou podporovány JBoss AS od verze 3.2 až po nejaktuálnější.

6.1.2 Projekt Spring Roo v Spring Tool Suite

Integraci Spring Roo do prostředí Eclipse řeší Spring Tool Suite, dostupné samostatně i ve formě modulu prostředí Eclipse. Kromě Spring Roo také obsahuje Apache Maven plugin, servletový kontejner Apache Tomcat a jiné moduly pro podporu vývoje za pomoci produktů společnosti SpringSource. Pro integraci JBoss AS je potřeba opět instalovat zásuvný modul JBoss Tools. Prostředí podporuje vytváření Spring Roo projektů, ovládání RAD nástroje je téměř shodné, jako ze standardního příkazového řádku.

6.1.3 Odstranění AspectJ souborů vytvořených se Spring Roo

Prostředí Eclipse v současnosti jako jediné obsahuje funkci převádějící aspektově orientovaný kód v mezitypových deklaracích na standardní Java EE kód, který je přesunut do odpovídajících tříd po jednotlivých metodách, celých aspektech, nebo v rámci celého projektu. Jedná se o funkci push-in, kterou poskytuje modul AJDT. Při jejím používání je doporučeno mít spuštěný nástroj Spring Roo. Použití má smysl v případech, kdy kód v pomocných souborech nevyhovuje potřebám vývojáře a vyžaduje editaci. Nevýhodou je snížení přehlednosti souboru nesoucího entitu.

Úplné odstranění Spring ROO z projektu spočívá v použití funkce push-in na úrovni celého projektu, smazání `aspectj-maven-plugin` položky a `org.springframework.roo.annotations` artefaktu z *pom.xml* souboru a odstranění `@Roo` anotací z .java souborů. Dále je potřeba odstranit AspectJ správu transakcí ze souboru *ApplicationContext.xml*.

Znovuzpřístupnění projektu nástroji Spring Roo je inverzí kroků z předchozího odstavce. Jelikož nástroj nespravuje kód vzniklý funkcí push-in, je doporučeno při znovuzpřístupnění projektu smazat (nebo převést do komentářů) prvky, které může Roo spravovat. Po smazání např. metod `toString()`, nebo `persist()`, převedených funkcí push-in, jsou automaticky obnoveny původní AspectJ soubory.

6.2 Podpora vývoje se Spring Roo v IntelliJ Idea

Podporu technologií Spring a Java EE obsahuje pouze placená verze. Do neplacené je možné Spring Roo projekt importovat jako Maven projekt. Úroveň podpory vývoje je srovnatelná s prostředím Eclipse z kapitoly 6.1.1 bez JBoss Tools a AJDT pluginu. Placená verze IntelliJ Idea obsahuje podporu frameworku Spring srovnatelnou s STS. Obsahuje v základu integraci aplikačních serverů včetně JBoss, nástroje Maven, jazyka AspectJ i Spring Roo.

6.3 Podpora vývoje se Spring Roo v Netbeans IDE

Netbeans IDE obsahuje podporu vývoje v Java EE 6, Spring Framework, včetně Spring Web MVC, obsahuje moduly Apache Maven, Hibernate. AspectJ soubory jsou rozpoznány, ale nejsou podporovány pokročilé funkce, jako automatické doplňování, či syntaktická kontrola. Prostředí nepodporuje aktuální verzi JBoss AS (pouze verze 4, 5, 6), ani integraci Spring Roo. Editor prostředí má doložené problémy s XML dokumenty (https://netbeans.org/bugzilla/show_bug.cgi?id=217342). Během testování hlásil chybu v elementu `<![CDATA[]]>` (příloha L). Toto ale nebránilo správnému sestavení aplikace.

6.4 Podpora vývoje s JBoss Forge v Eclipse

Pro vývoj v JBoss Forge je zásadním prvkem Maven Integration plugin a JBoss Tools plugin. Součástí JBoss Tools je mimojiné integrace JBoss Forge i podpora wysiwyg (what you see is what you get, neboli průběžné zobrazení tvořené stránky) vývoje ve frameworku JSF. Stávající projekt jde otevřít jako Maven projekt, sestavit a nasadit na server bez problémů. Nový projekt lze založit ve formě některého z Maven archetypů, nebo z integrovaného JBoss Forge, a projekt následně importovat.

6.5 Podpora vývoje s JBoss Forge v IntelliJ Idea

IntelliJ Idea nepodporuje v žádné verzi integraci nástroje JBoss Forge a v bezplatné verzi nepodporuje ani Java EE technologie. Opět lze projekt importovat jako Maven projekt a využít alespoň integrace nástroje Apache Maven. Placená verze naopak plně podporuje vývoj v Java EE, obsahuje i JBoss AS 7 modul. Nastavení projektu je potřeba pozměnit, označit použité technologie a přiřadit v nastavení prostředí typ souboru **.xhtml* mezi soubory obsahující JSP a JSF kód. Automatická detekce konfigurace projektu zde neuspěla.

6.6 Podpora vývoje s JBoss Forge v Netbeans IDE

Prostředí nepodporuje integraci JBoss Forge, ani nejnovější verze JBoss AS. Podporuje vývoj ve všech v této práci použitých technologiích Java EE. Projekt lze otevřít a sestavit. Jako jediné prostředí odhalilo syntaktickou chybu ve vygenerovaných JSF **.xhtml* souborech. Jednalo se o atribut `maxlength` u tagu `<h:inputText>`, vygenerovaný společně s Hibernate validačními omezeními (příloha M).

6.7 Shrnutí

Všechna zmíněná prostředí obsahují modul nástroje Apache Maven. Projekt vytvořený se Spring Roo jsou schopné otevřít a sestavit. Kromě neplacené verze IntelliJ Idea podporují všechna prostředí Java EE 6, Spring Framework, Seam Framework, Hibernate, Spring MVC, JSF 2, tedy technologie použité v této práci.

Nejpokročilejší podporu vývoje s nástroji JBoss Forge a Spring Roo obsahuje prostředí Eclipse IDE díky rozšíření STS a modulu JBoss Tools. Podporuje oba RAD nástroje i aplikační server JBoss AS 7.

Podpora	Eclipse IDE	IntelliJ Idea	Netbeans IDE
Apache Maven	ano, formou modulu	ano, integrován	ano, integrován
JBoss AS	ano, formou modulu	jen v placené verzi	jen starší verze JBoss AS
Spring Roo	ano, v STS	jen v placené verzi	ne
JBoss Forge	ano, formou modulu	ne	ne

Tabulka 6.1: Porovnání podpory ve vývojových prostředích

Placená verze IntelliJ Idea má předinstalován modul Spring Roo i JBoss AS 7, nepodporuje však JBoss Forge. Neplacená verze vývoj enterprise aplikací nepodporuje. Netbeans IDE nepodporuje žádný z RAD nástrojů, ani aktuální verzi serveru JBoss AS.

Prostředí Netbeans IDE a IntelliJ Idea umožňují spustit nástroje skrze integrovaný příkazový řádek, nebo jako externí nástroj běžící v konzoli prostředí. Použití této funkce se neosvědčilo, chování spuštěných nástrojů je chybné.

Kapitola 7

Další možnosti uplatnění nástrojů

7.1 Použití v již vytvořených projektech

Žádný z obou nástrojů není přímo určen k práci na již započatých Java EE projektech. Povahou se jedná o nástroje pro tvorbu prototypů aplikací, tedy v počátečních fázích vývoje. Tvůrci Spring Roo proto doporučují zvážit ukončení původního projektu a začít nový s RAD nástrojem [3]. Přesto bylo porovnáno i uplatnění nástrojů v již započatém projektu.

Nevýhodou takové konverze je nemožnost využít automatické konfigurace projektu, u Spring Roo nutnost manuálně přizpůsobit závislosti a konfiguraci RAD nástroji.

Oba nástroje dokáží také vytvořit prototyp aplikace na základě relační databáze. Reverzní inženýrství bylo otestováno na vzorové H2 databázi (stránky projektu H2 http://www.h2database.com/html/main.html), obsahující model z kapitoly 5. Databáze byla v základním nastavení, ovladač databáze byl součástí archivu staženého z jejích domovských stránek.

7.1.1 Spring Roo ve Spring projektech

Pro instalaci Spring Roo do Apache Maven projektu je potřeba provést kroky z posledního odstavce kapitoly 6.1.3 a doplnit konfiguraci do souboru *pom.xml* a *ApplicationContext.xml* podle, například, prázdného Spring Roo projektu. Pokud je použito vývojové prostředí Eclipse STS, jeho soubor *.project* by měl obsahovat položky ukázané v příloze N. Důležité je také vložit konfigurační soubory Spring Frameworku do stejně strukturovaných a pojmenovaných adresářů, jako ve Spring Roo vygenerovaném projektu.

Spuštěný nástroj poté doplní POJO objekty o mezitypové deklarace podle `@Roo` anotací, které do nich byly vloženy. Bez vhodně nastavené jednotky perzistence a JPA ale nelze již provádět další operace. Například v případech, kdy je nastavení jednotky perzistence v **.java* souboru, nebo u projektu bez podpory JPA, nelze ani jeden z porovnávaných RAD nástrojů použít.

7.1.2 JBoss Forge v Java EE projektech

Prekvizitou použití JBoss Forge v projektu je stejně jako ve Spring Roo správa projektu pomocí Apache Maven. Tvorba modelu vyžaduje nastavenou jednotku perzistence a JPA v souboru *persistence.xml*. V souboru *pom.xml*, ani v souborech entit, nejsou pro jeho fungování vyžadovány žádné úpravy.

7.1.3 Spring Roo a scaffolding z databáze

Před samotným reverzním inženýrstvím databáze je potřeba vytvořit projekt a nastavit perzistenci podle vzoru z kapitoly 5.

V době psaní práce byl díky chybě v repozitáři Spring Roo (dokumentace chyby

https://jira.springsource.org/browse/ROO-3184) nefunkční systém instalace doplňků. Toto brání instalaci jdbc ovladače a dalším krokům z této části práce.

V knize „Getting Started with Roo" [21] a „Spring Roo in Action" [22] je však uveden následující postup. Pro vytvoření modelu podle schématu databáze slouží příkaz `database reverse engineer`. V parametrech příkazu lze vybrat databázi, které tabulky databáze pojmout (nebo vynechat) a do jaké složky projektu model vytvořit. Po zadání příkazu je uživatel vyzván k instalaci modulu s ovladačem databáze zvolené při nastavování perzistence, s tímto je mu prezentován seznam dostupných modulů s požadovaným ovladačem. Tento modul nainstaluje příkaz addon install --searchResultId a číslo vybraného modulu. Vlastnosti připojení k databázi (URL, přihlašovací údaje) nastaví příkaz `database properties set`, který edituje soubor *src/main/resources/META-INF/spring/database.properties*. Entity jsou vygenerovány do souborů mezitypových deklarací *$NAZEV_ENTITY_Roo_DbManaged.aj*.

7.1.4 JBoss Forge a scaffolding z databáze

I zde jsou prekvizitou reverzního inženýrství první dva (případně tři) kroky tvorby aplikace popsané v kapitole 5.2.1. Následuje instalace zásuvného modulu `forge install-plugin hibernate-tools` a příkaz `generate-entities` s parametry použité databáze, ovladačem jdbc pro komunikaci s databází, URL databáze, jménem uživatele. Další kroky kopírují kapitolu 5 (instalace prezentační technologie, vygenerování GUI).

7.2 Porovnání možností testování

Testování je nezbytnou součástí vývoje jakékoliv aplikace. V jazyce Java jsou základem testování jednotkové testy (unit tests), neboli izolované testy menších částí aplikace, zejména tříd a metod. Pro usnadnění tvorby jednotkových a integračních testů vznikl framework JUnit. Integrační testy zkoumají chování jednotek ve vzájemných vazbách v rámci systému a při práci se zdroji (např. databáze), proto vyžadují běhové prostředí (např. aplikační server). Podpora testů byla následně přidána do vývojových prostředí a nástrojů pro tvorbu buildů. Dalším druhem testů jsou funkcionální testy simulující požadavky klieta externími nástroji (např. emulátor prohlížeče). Kapitola 7.2 se bude zabývat možnostmi, které JBoss Forge a Spring Roo nabízí při tvorbě testů. Veškeré testy jsou v projektech Apache Maven tvořeny ve složce *src/test*.

7.2.1 Tvorba testů se Spring Roo

Spring Roo podporuje tvorbu jednotkových testů. Jedná se o příkazy `test stub` a `test mock`. Stub je kostrou testu, který zjednodušeným způsobem implementuje rozhraní volané testovaným objektem. Účelem testu je zkoumat interakci mezi dvěma objekty a jejich rozhraními. Jeho vlastnosti a chování jsou implementovány programátorem a plně pod jeho kontrolou. Stejný cíl mají i mock objekty, které

implementují jenom potřebnou část rozhraní. Zavolání příkazu `test mock` na entitu vygeneruje soubor (šablonu testu) s jednoduchým testem metody sčítající instance entity.

Spring Roo vygeneruje integrační testy, pokud je při vytváření entit zadán přepínač `--testAutomatically`, nebo samostatně příkazem, například `test integration --entity .model.Customer` . Tento příkaz vytvoří soubory *CustomerIntegrationTest.java*, příslušný aspekt (obsahující testy), *CustomerDataOnDemand.java* a příšlušný aspekt (poskytující a vytvářející testovací data).

Data on Demand (DoD) je spring Roo framework, součást specifikace Spring Roo. Jeho úkolem je vytváření dat pro integrační testy. Vygenerovaný DoD objekt vytvoří 10 instancí objektu. Jeho hlavními metodami jsou `getTransientEntity()`, `getSpecificEntity()` a textttgetRandomEntity(). Tyto metody vrací (a případně vytváří) instanci požadované entity. První jmenovaná metoda nevyužívá kontejner, lze ji použít i pro jednotkové testy.

Integrační testy jsou vygenerovány pro CRUD metody entity. Generátor testů nereflektuje validační omezení, pokud jsou aplikována, a testy se musí manuálně upravit.

S nastavením perzistence, upraveným pro server JBoss AS, dochází při integračních testech k chybě objektu `EntityManager`. Nastavení aplikace používá perzistenční jednotku serveru. Ta je ve fázi běhu testů nedostupná, protože při běhu JUnit testů není aplikace nasazena na server. Možným řešením je vytvoření druhé jednotky perzistence ve složce */test/META-INF* s přístupem k jinému zdroji dat. Spustit jdou testy ve výjovém prostředí, nebo při sestavování aplikace nástrojem Apache Maven.

Funkcionální testy vykonává framework Selenium. Příkaz `selenium test --controller .web.JmenoKontroleru` vygeneruje test pro zadaný kontroler. Účelem těchto testů je kontrola návratových hodnot kontrolerů, testování případů užití, nebo testování vytížení [22].

7.2.2 Tvorba testů v JBoss Forge

Jboss Forge pro účely testování používá framework Arquillian, ten je dostupný ve formě modulu. Arquilian zahrnuje podporu jednotkových, integračních a funkcionálních (Selenium) testů. Jeho vlastnosti jsou kromě sjednocení testovacích technologií a přístupů také zjednodušení konfigurace, portabilita mezi kontejnery a aplikačními servery, podpora ve vývojových prostředích a nástrojích pro tvorbu buildů. Z pohledu implementace se jedná o testy JUnit s anotacemi konfigurujícími framework Arquillian. Základní anotace jsou `@RunWith Arquillian`, která předá zodpovědnost nad vykonáním testu, dále `@Deployment`, která uvozuje metodu, deklarující vlastnosti archivu s integračními testy, nasazeného na cílový testovací kontejner či aplikační server. Testy jsou zkompilovány do balíčku, nasazeny a spuštěny. Za tímto účelem Arquillian používá mechanismus pro tvorbu archivů JBoss Shrinkwrap (http://www.jboss.org/shrinkwrap).

Základy použití JBoss Forge a frameworku Arquillian jsou popsány na stránkách JBoss Forge [23]. Po instalaci rozšíření je do POM souboru přidán profil pro běh na zvoleném kontejneru příkazem `arquillian setup`. Šablona testu je vygenerována příkazem `arquillian create-test`. Vygenerovaný soubor obsahuje metodu `@Deployment` a jednoduchou zkoušku úspěšného nasazení testu na kontejner. Metoda `@Deployment` vyžaduje doplnění objektů závislých na testovaném. Aby test proběhl, musí být jeho profil (vygenerovaný v POM souboru) aktivní. Pokud má být otestována vrstva perzistence, je nutné také nastavit v metodě `@Deployment` cestu ke konfiguraci perzistence (*persistence.xml*), vložit do testu `EntityManager` objekt a další kroky podle vlastností testu [24].

7.3 Nasazení na platformě OpenShift

Poslední kapitola se zabývá nasazením na PaaS (Platform as a Service) platformě OpenShift a porovnání aplikací v prohlížeči na PC i na přenosných zařízeních. OpenShift je PaaS cloud službou společnosti Red Hat. Jejím účelem je poskytnout zákazníkovi prostředí pro běh jeho aplikací, tedy hardwarové, softwarové prostředky a jejich správu. Podporuje nasazení internetových aplikací z širokého spektra jazyků a jejich frameworků. Obě vzorové aplikace byly nasazeny na platformě OpenShift, ale neplacená verze není výkonově robustní, proto se aplikace z následujících odkazů někdy nenačte: http://rooshop-ondraen1.rhcloud.com/rooshop/ a http://forgeshop-ondraen1.rhcloud.com/forgeshop/faces/index.xhtml.

7.3.1 Vytvoření projektu, nasazení aplikace

Projekt je uložen na serveru Openshift v git repozitáři. Lze jej vytvořit více způsoby: skrze webové rozhraní služby, nástrojem rhc (příkaz `rhc app create`), skrze JBoss Developer Studio IDE a JBoss Tools plugin pro Eclipse IDE.

Repozitář je možné používat standardně pro vývoj projektu, nebo druhou možností je nahrát pouze samostatný balíček projektu a projekt vyvíjet odděleně. Projekt je po každé změně repozitáře automaticky sestaven a nasazen.

Rhc je nástroj určený pro správu projektů na službě Openshift, běžící v příkazové řádce. Po založení účtu a vytvoření nového projektu řídí další kroky instrukce souboru *README* vygenerovaném v domovské složce projektu. Ty popisují možné způsoby nasazení a dávají informace o použití nástroje. Před nasazením balíčku je třeba odstranit soubory aplikace a vložit kopii balíčku projektu do složky *deployments*. Posloupnost příkazů `commit` a `push` nástroje git propaguje změny na vzdáleném repozitáři serveru a projekt je v řádu desítek vteřin nasazený a přístupný z internetu.

7.3.2 Aplikace v prohlížeči a na přenosném zařízení

Aplikace nasazené na službě Openshift byly otestovány validátorem W3C na desktop i přenosné zařízení. Na stránkách aplikace forgeshop při HTML5 testu nebyly nalezeny chyby. Na XHTML stránkách rooshop, podobně jako při jejich HTML 5 validaci, byly nalezeny nepodporované atributy. Při testu vhodnosti stránek pro mobilní zařízení ani jedna z obou aplikací neprošla bezchybně. Při testování v prohlížeči Firefox (verze 19.0.2), Chrome (verze 25.0.1364.160) a Windows Explorer 9 se neprojevily žádné nedostatky ani v jediném případě. Podobně při používání stránek na zařízení se systémem Android 4.0 s prohlížečem Opera Mini a Firefox.

Kapitola 8

Závěr

Práce ukázala, že v počátečním stádiu tvorby projektu JBoss Forge a Spring Roo dokáží výrazně urychlit vývoj. Oba nástroje lze označit za podobně efektivní co do počtu příkazů a znaků zadaných nástroji při tvorbě prototypu aplikace. Množství vygenerovaného kódu je na straně Spring Roo výrazně vyšší kvůli struktuře Spring aplikací. Většina kódu modelu aplikace vygenerované Spring Roo je v souborech mezitypových deklarací. Tyto AspectJ soubory zlepšují přehlednost vygenerovaného kódu, rozdělují jej na část spravovanou programátorem od té spravované nástrojem. Mezitypové deklarace, pokud nevyhovují, nebo kvůli manuální editaci, lze vložit do Java souboru, na který jsou vázané. Při vytváření vzorové aplikace bylo zjištěno, že pro nasazení na aplikační server JBoss AS vyžaduje program, vygenerovaný Spring Roo, úpravy v konfiguraci perzistence, další úpravy jsou nutné pro potřeby integračních testů. JBoss Forge v porovnání se Spring Roo nezanechává žádné stopy, kód je po vygenerování celý přístupný programátorovi a dále není nástrojem spravován. Dodatečné úpravy před nasazením nejsou potřeba.

Zkoumání podpory nástrojů ve vývojových prostředích ukázalo, že integraci JBoss Forge nabízí pouze prostředí Eclipse ve formě rozšíření JBoss Tools. Spring Roo je obsažen v Eclipse STS a je také součástí placené verze IDE IntelliJ Idea, není však integrován v prostředí Netbeans. Aplikační server JBoss AS 7 podporují pouze prostředí Eclipse (JBoss Tools) a placené verze IDE IntelliJ Idea. Všechna testovaná prostředí obsahují nástroj Maven.

Pokročilejšími funkcemi pro tvorbu testů je osazeno Spring Roo. Jeho součástí jsou generátory šablon pro jednotkové testy, plnohodnotných integračních testů pro CRUD funkcionalitu, a plnohodnotných testů uživatelského rozhraní. Součástí nástroje je i framework pro tvorbu testovacích dat. Jboss Forge generuje testy s pomocí frameworku Arquillian pouze ve formě šablon.

Pro použití na již započatém Apache Maven projektu Spring Roo vyžaduje úpravy POM souboru, souboru *ApplicationContext.xml*, nastavenou jednotku perzistence a JPA v souboru *persistence.xml*. JBoss Forge stačí pro základní použití pouze nastavení jednotky perzistence a JPA. Vytvoření aplikace na základě databázového modelu se podařilo pouze v případě JBoss Forge, Spring Roo v současnosti trpí chybou ve správě doplňků, které jsou k provedení této operace nutné.

Obě vzorové aplikace se podařilo nasadit na PaaS službu Openshift a otestovat validátorem W3C. Na přenosném zařízení s operačním systémem Android, ani ve webovém prohlížeči na osobním počítači nebyly zaznamenány problémy, jako například nepodporované prvky. Validátor W3C pro mobilní zařízení odhalil nedostatky obou aplikací, ani jedna z aplikací pak neprošla bez chyby XHTML validací, aplikace

vygenerovaná JBoss Forge splňuje specifikaci HTML5, Spring Roo ne.

Oba nástroje zastávají odlišnou filozofii. Spring Roo je úzce spjatý s platformou Spring, dokáže automaticky reagovat na manuální úpravy modelu aplikace úpravou kódu kontrolerů a kódu prezentačního, je schopen i automaticky aktualizovat závislosti v projektu. V současnosti existuje několik desítek modulů rozšiřujících funkce Spring Roo včetně modulů několika odlišných prezentačních technologií. Výhodou Spring Roo je širší vývojářská a uživatelská základna, s čímž souvisí kvalitní dokumentace k nástroji a technologiím, které používá. JBoss Forge je koncipován jako nástroj nezávislý na platformě, ovšem dnes jsou dostupné pouze funkce pro vývoj za použití produktů firmy Red Hat a standardu Java EE 6. Narozdíl od Spring Roo neobsahuje automatickou správu kódu. Dokumentace k JBoss Forge je v tuto chvíli ve stádiu tvorby kvůli menšímu počtu zúčastněných osob a celkovému mládí projektu.

Díky vlivu RAD nástrojů na zvýšení konkurenceschopnosti podnikových Java projektů a růstu komunity kolem nich, lze v následujících letech očekávat lepší podporu Spring Roo a JBoss Forge i mimo prostředí Eclipse a jejich postupné přijetí jako plnohodnotného nástroje pro zrychlení vývoje. S délkou doby, kterou jsou tyto nástroje dostupné také souvisí množství chyb a nedostatků. To se bude časem dále snižovat. Poroste i množství funkcí a doplňků pokrývající potřeby širší komunity. Díky současným požadavkům trhu na rychlé dodání aplikací budou mít RAD nástroje stále větší vliv na vývoj samotných aplikací i standardů, podle kterých jsou aplikace tvořeny.

Literatura

[1] Red Hat Middleware, LLC. Seam 3: Bundled Reference Guide [online]. c2011, [cit. 2012-12-13]. Dostupné na World Wide Web: <http://docs.jboss.org/seam/3/latest/reference/en-US/pdf/bundled_master.pdf>.

[2] Johnson, Rod, et al. Spring Framework Reference Documentation [online]. c2012, [cit. 2012-12-13]. Dostupné na World Wide Web: <http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/>.

[3] VMware, Inc. Spring Roo - Reference Documentation: 1.2.2.RELEASE [online]. c2012, [cit. 2012-12-13]. Dostupné na World Wide Web: <http://static.springsource.org/spring-roo/reference/pdf/spring-roo-docs.pdf>.

[4] Srirangan. Apache Maven 3 Cookbook. 1. ed. Birmingham: Packt Publishing Ltd, 2011. 224 p. ISBN 978-1-849512-44-2.

[5] Oracle Corporation. Introduction to Contexts and Dependency Injection for the Java EE Platform [online]. c2013, [cit. 2013-4-25]. Dostupné na World Wide Web: <http://docs.oracle.com/javaee/6/tutorial/doc/giwhb.html>.

[6] King, Gavin, et al. Weld - JSR-299 Reference Implementation [online]. c2010, [cit. 2012-12-13]. Dostupné na World Wide Web: <http://docs.jboss.org/weld/reference/1.0.1-Final/en-US/html/intro.html>.

[7] Yii Software LLC. Best MVC Practices [online]. c2010, [cit. 2012-12-13]. Dostupné na World Wide Web: <http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices>.

[8] Oracle Corporation. JavaServer Pages Technology [online]. [cit. 2012-12-13]. Dostupné na World Wide Web: <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>.

[9] Murray, Steven. Designing JSF Applications - A Storyboard Approach [online]. c2008, [cit. 2012-12-13]. Dostupné na World Wide Web: <http://www.jsfcentral.com/articles/storyboard_2.html>.

[10] Fowler, Martin. Patterns of Enterprise Application Architecture. 1. ed. Addison-Wesley Professional, 2002. 560p. ISBN 978-0321127426.

[11] Oracle Corporation. JAR File Specification [online]. c2010, [cit. 2013-4-25]. Dostupné na World Wide Web: <http://docs.oracle.com/javase/1.5.0/docs/guide/jar/jar.html>.

[12] Wikipedia. Rapid application development [online]. Last revision 14th of May 2013, [cit. 2013-5-14]. Dostupné na World Wide Web: <http://en.wikipedia.org/wiki/Rapid_application_development>.

[13] Jonathan Bartlett. The art of metaprogramming, Part 1: Introduction to metaprogramming [online]. c2005, [cit. 2013-2-14]. Dostupné na World Wide Web: <http://www.ibm.com/developerworks/library/l-metaprog1/index.html>.

[14] Django Software Foundation. Django Documentation: Core Philosophies [online]. c2013, [cit. 2013-2-14]. Dostupné na World Wide Web: <https://docs.djangoproject.com/en/dev/misc/design-philosophies/>.

[15] Django Software Foundation. Django Documentation: Middleware [online]. c2013, [cit. 2013-2-14]. Dostupné na World Wide Web: <https://docs.djangoproject.com/en/dev/topics/http/middleware/>.

[16] Oracle Corporation. The Java EE 5 Tutorial: Web Archives [online]. c2010, [cit. 2013-2-14]. Dostupné na World Wide Web: <http://docs.oracle.com/javaee/5/tutorial/doc/bnadx.html>.

[17] Guillo Zambon. Beginning Jsp, Jsf and Tomcat. 2. ed. New York: Apress, 2012. 436 p. ISBN 978-1-4302-4624-4.

[18] Oracle Corporation. The Java EE 5 Tutorial: Using Scopes [online]. c2010, [cit. 2013-2-14]. Dostupné na World Wide Web: <http://docs.oracle.com/javaee/6/tutorial/doc/gjbbk.html>.

[19] Keith Donald et al. Spring Web Flow Reference Guide [online]. c2013, [cit. 2013-2-14]. Dostupné na World Wide Web: <http://static.springsource.org/spring-webflow/docs/2.3.x/reference/html/>.

[20] Oracle Corporation. The Java EE 5 Tutorial: JavaServer Pages Documents [online]. c2010, [cit. 2013-2-14]. Dostupné na World Wide Web: <http://docs.oracle.com/javaee/5/tutorial/doc/bnajo.html>.

[21] Long, Josh, Steve Mayzak. Getting Started with Roo. 1. ed. O'Reilly Media, 2011. 64p. ISBN 978-1449307905.

[22] Rimple, Ken, Srini Penchikala. Spring Roo in Action. Manning Publications, 2012. 408p. ISBN 978-1935182962.

[23] Red Hat, Inc. JBoss Forge documentation [online]. c2013, [cit. 2013-4-25]. Dostupné na World Wide Web: <http://forge.jboss.org/docs/index.html>.

[24] Red Hat, Inc. Arquillian Reference Guide [online]. c2011, [cit. 2013-4-25]. Dostupné na World Wide Web: <https://docs.jboss.org/author/display/ARQ/Reference+Guide>.

[25] Oracle Corporation. Differences between Java EE and Java SE [online]. c2012, [cit. 2013-5-10]. Dostupné na World Wide Web: <http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html>.

[26] Johnson, Rod. Expert One on One J2ee Design and Development. Peer Information, 2002. 750p. ISBN 1861007841.

[27] Juneau, Josh et al. The Definitive Guide to Jython [online]. c2010, [cit. 2013-5-10]. Dostupné na World Wide Web: <http://www.jython.org/jythonbook/en/1.0/>.

Přílohy

A) Posloupnost příkazů vedoucí k vytvoření aplikace - Spring Roo

1. project --topLevelPackage com.rooshop
2. jpa setup --database HYPERSONIC_PERSISTENT --provider HIBERNATE --jndiDataSource java:/jboss/datasources/ExampleDS
3. entity jpa --class ~.model.Customer
4. field string --fieldName lastName --notNull –sizeMin 1 --sizeMax 30
5. field string --fieldName firstName --notNull --sizeMin 1 --sizeMax 30
6. entity jpa --class ~.model.CartOrder
7. field boolean --fieldName inCart
8. field date --fieldName dateOfOrder --type java.util.Date --notNull
9. focus --class ~.model.Customer
10. field set --fieldName cartOrders --type ~.model.CartOrder --cardinality ONE_TO_MANY --mappedBy customer
11. focus --class ~.model.CartOrder
12. field reference --fieldName customer --type ~.model.Customer --cardinality MANY_TO_ONE
13. entity jpa --class ~.model.Company
14. field string --fieldName name --notNull --sizeMin 1 --sizeMax 255
15. field set --fieldName customers --type ~.model.Customer --cardinality ONE_TO_MANY --mappedBy company
16. focus --class Customer
17. field reference --fieldName company --type ~.model.company --cardinality MANY_TO_ONE
18. entity jpa --class ~.model.LineItem

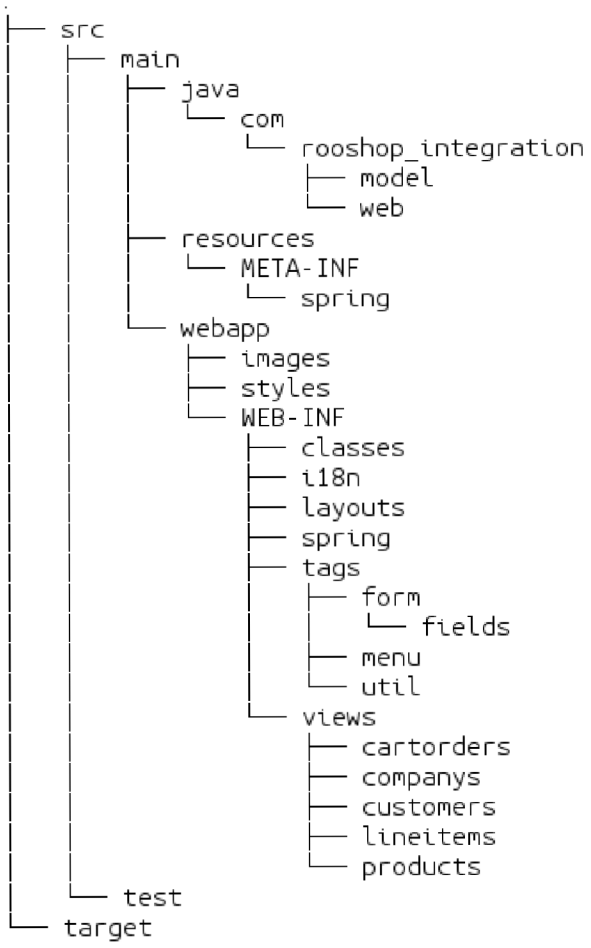
19. field number --fieldName quantity --type int
20. focus --class CartOrder
21. field set --fieldName lineItems --type ~.model.LineItem --cardinality ONE_TO_MANY --mappedBy cartOrder
22. focus --class LineItem
23. field reference --fieldName cartOrder --type ~.model.CartOrder --cardinality MANY_TO_ONE
24. entity jpa --class ~.model.Product
25. field string --fieldName name --notNull --sizeMin 1 --sizeMax 30
26. field string --fieldName description --sizeMax 1000
27. field number --fieldName price --type double
28. field set --fieldName lineItems --type ~.model.LineItem --cardinality ONE_TO_MANY --mappedBy product
29. focus --class LineItem
30. field reference --fieldName product --type ~.model.Product --cardinality MANY_TO_ONE
31. web mvc setup
32. web mvc all --package ~.web
33. perform package
34. exit

B) Posloupnost příkazů vedoucí k vytvoření aplikace: JBoss Forge

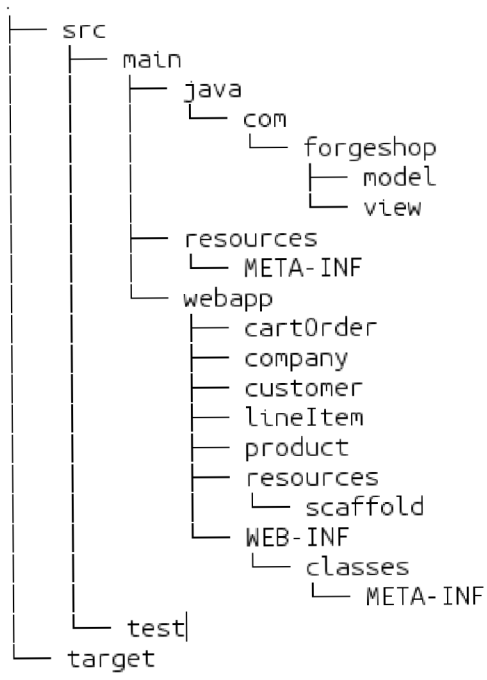
1. new-project --named forgeshop --topLevelPackage com.forgeshop
2. persistence setup --provider HIBERNATE --container JBOSS_AS7
3. validation setup --provider HIBERNATE_VALIDATOR
4. entity --named Customer --package com.forgeshop.model
5. field string --named lastName
6. constraint NotNull --onProperty lastName
7. constraint Size --onProperty lastName --min 1 --max 30 --message "Maximum 30 znaku, minimum 1."
8. field string --named firstName
9. constraint NotNull --onProperty firstName
10. constraint Size --onProperty firstName --min 1 --max 30 –message "Maximum 30 znaku, minimum 1."
11. entity --named CartOrder –package com.forgeshop.model
12. field boolean --named inCart
13. field temporal --named dateOfOrder --type java.util.Date
14. constraint NotNull --onProperty dateOfOrder
15. cd ../Customer.java
16. field oneToMany --named cartOrders --fieldType com.forgeshop.model.CartOrder --inverseFieldName customer
17. entity --named Company --package com.forgeshop.model
18. field string --named name
19. constraint NotNull --onProperty name
20. constraint Size --onProperty name --min 1 --max 255 --message "Maximum 255 znaku, minimum 1."
21. field oneToMany --named customers --fieldType com.forgeshop.model.Customer --inverseFieldName company
22. entity --named LineItem --package com.forgeshop.model
23. field int --named quantity

24. field manyToOne --named cartOrder --fieldType com.forgeshop.model.CartOrder --inverseFieldName lineItems
25. entity --named Product field string --named name
26. constraint NotNull --onProperty name
27. constraint Size --onProperty name --min 1 --max 30 --message "Maximum 30 znaku, minimum 1."
28. field string --named description
29. constraint Size --onProperty description --max 1000 --message "Maximum 1000 znaku."
30. field number --named price --type java.lang.Double
31. field oneToMany --named lineItems --fieldType com.forgeshop.model.LineItem --inverseFieldName product
32. scaffold setup
33. scaffold indexes
34. scaffold from-entity com.forgeshop.model.*
35. mvn package
36. exit

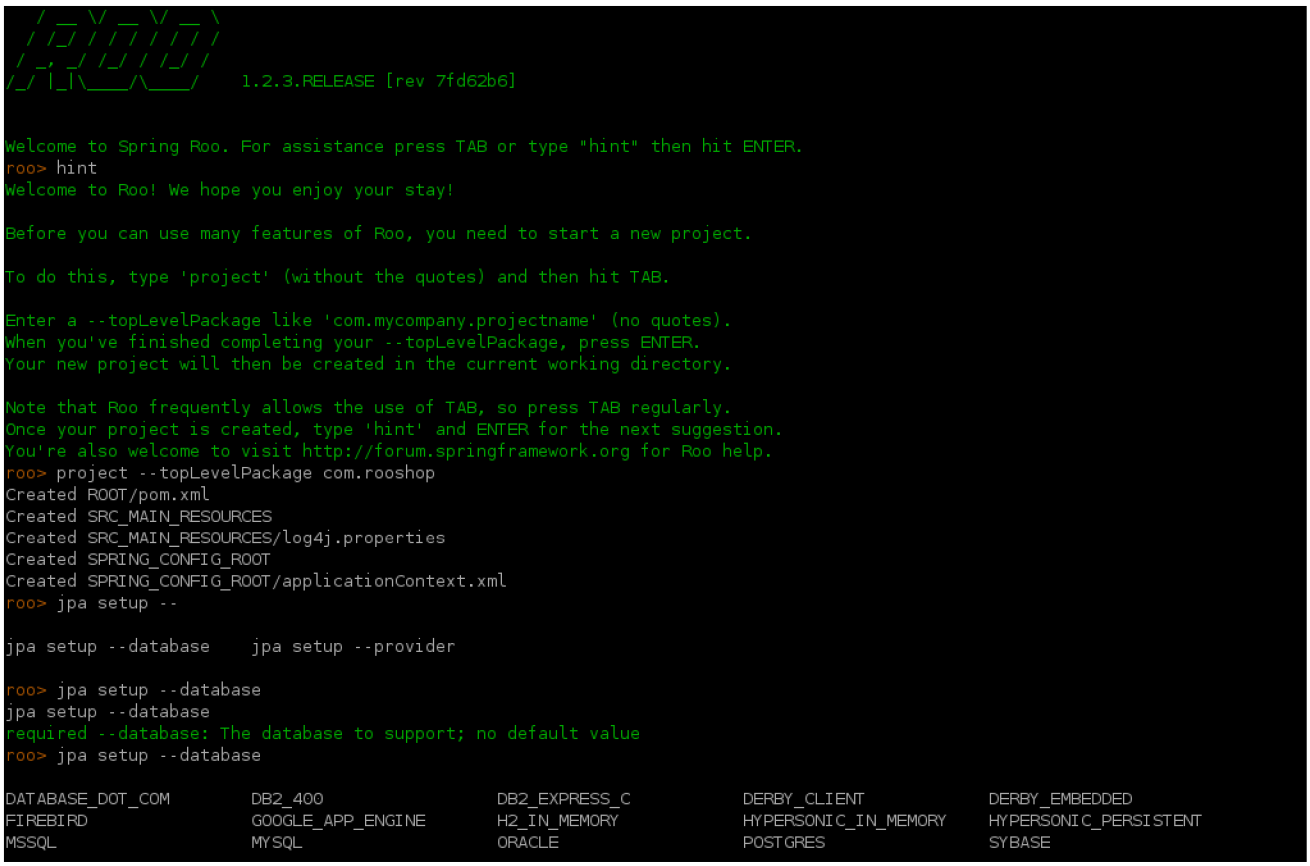
C) Struktura vzorové aplikace vygenerované ve Spring Roo



D) Struktura vzorové aplikace vygenerované v JBoss Forge



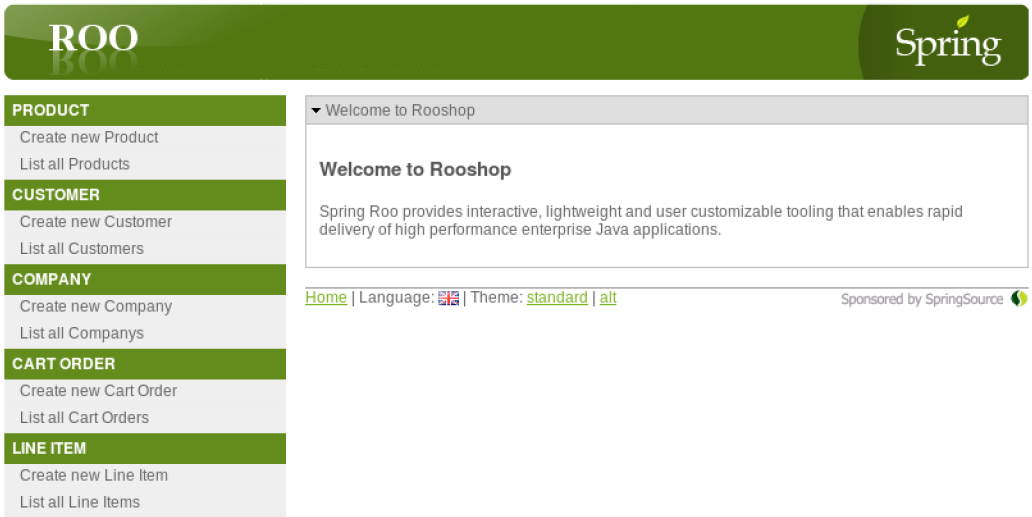
E) Konzole nástroje Spring Roo, nápověda, vytvoření projektu, perzistence



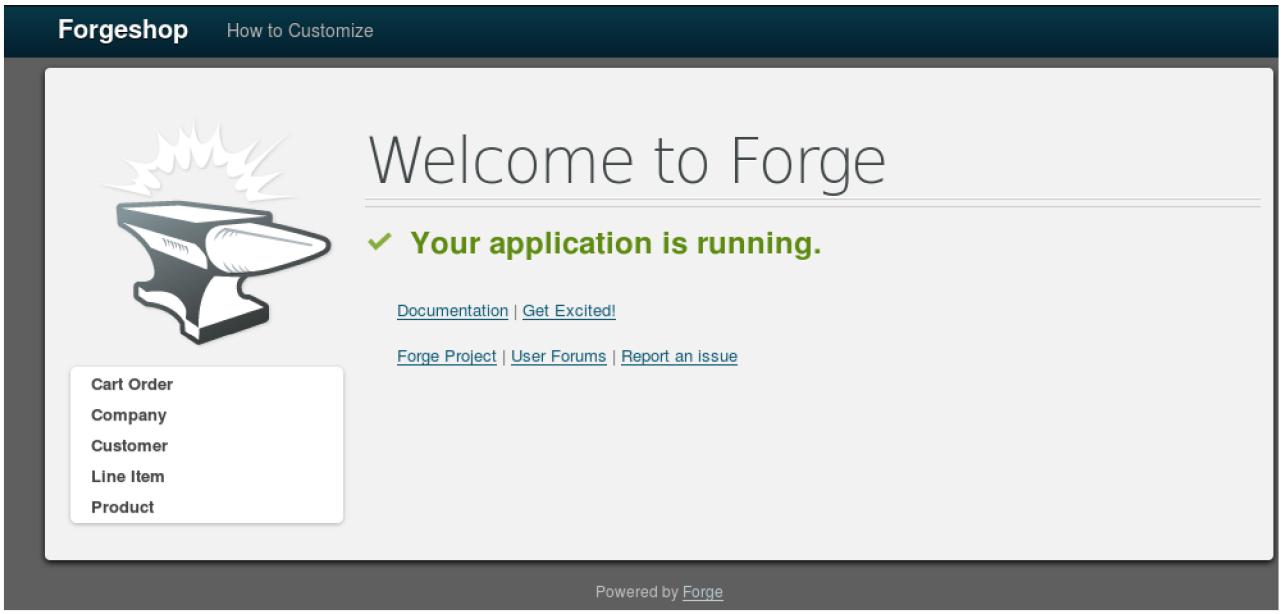
F) Konzole nástroje JBoss Forge, nápověda, vytvoření projektu, perzistence

```
ondraen@ondraen-ThinkPad:~/Documents/forge1/ScreenshotVersion$ forge
    _____
   |  ___|__  _ __ __ _  ___
   | |_ / _ \| `__/ _` |/ _ \  \\
   |  _| (_) | | | (_| |  __/  //
   |_|  \___/|_|  \__, |\___|
                     |___/

JBoss Forge, version [ 1.2.1.Final ] - JBoss, by Red Hat, Inc. [ http://forge.jboss.org ]
[no project] ScreenshotVersion $ new-project --named forgeshop --t
--topLevelPackage    --type
[no project] ScreenshotVersion $ new-project --named forgeshop --topLevelPackage com.forgeshop
 ? Use [/home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop] as project directory? [Y/n] y
***SUCCESS*** Created project [forgeshop] in new working directory [/home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop]
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop/pom.xml
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop/src/main/java
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop/src/test/java
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop/src/main/resources
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop/src/test/resources
Wrote /home/ondraen/Documents/forge1/ScreenshotVersion/forgeshop/src/main/java/com/forgeshop
[forgeshop] forgeshop $ persistence setup --provider HIBERNATE --container
JBOSS_AS6         JBOSS_AS7         JBOSS_EAP6        GLASSFISH_3       WEBLOGIC_12C      CUSTOM_JDBC       CUSTOM_JTA
CUSTOM_NON_JTA    TOMEE
[forgeshop] forgeshop $ persistence setup --provider HIBERNATE --container CUSTOM_JTA --
--provider-version    --database            --jndiDataSource      --jdbcDriver          --jdbcURL
--jdbcUsername        --jdbcPassword        --jta                 --named
[forgeshop] forgeshop $ persistence setup --provider HIBERNATE --container CUSTOM_JTA --database
MYSQL               ORACLE              DERBY               DB2                 POSTGRES            DEFAULT
DB2_AS400           DB2_OS390           MYSQL5_INNODB       MYSQL_INNODB        MYSQL5_ISAM         MYSQL_ISAM
ORACLE_9I           ORACLE_10G          SYBASE              SYBASE_ANYWHERE     SQL_SERVER          SAP_DB
INFORMIX            HSQLDB              INGRES              PROGRESS            MCKOI               INTERBASE
POINTBASE           FRONTBASE           FIREBIRD            HSQLDB_IN_MEMORY    ORACLE_11G          ACCESS
[forgeshop] forgeshop $ persistence setup --provider HIBERNATE --container CUSTOM_JTA --database
```

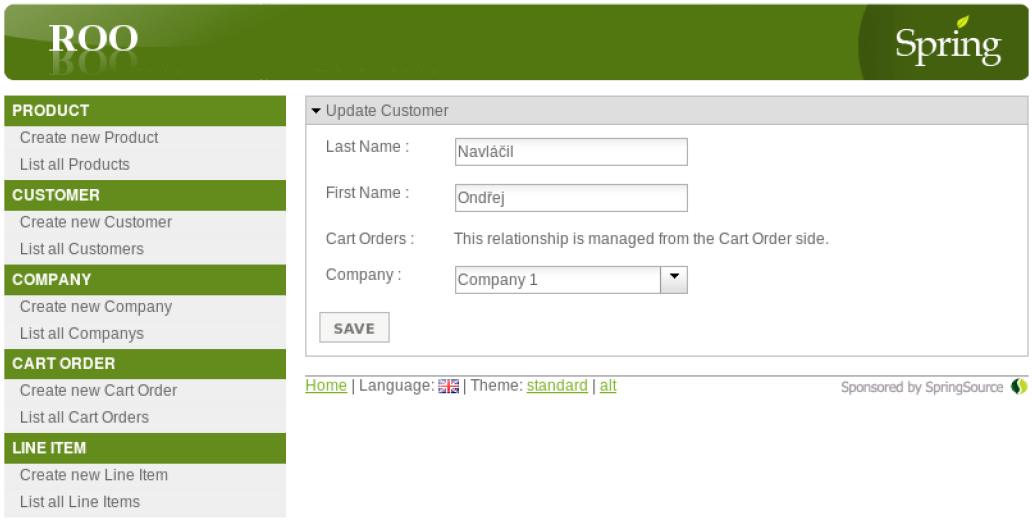
G) Úvodní stránka vzorové aplikace: Spring Roo



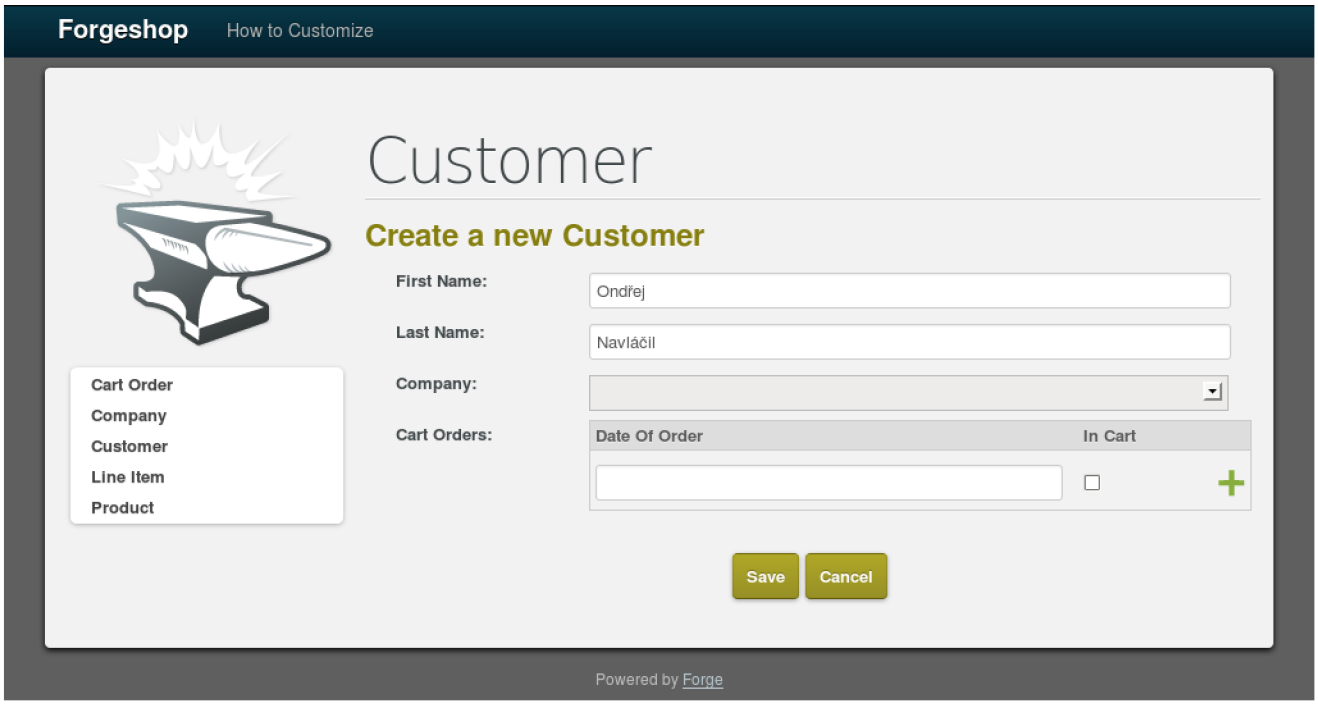
H) Úvodní stránka vzorové aplikace: JBoss Forge



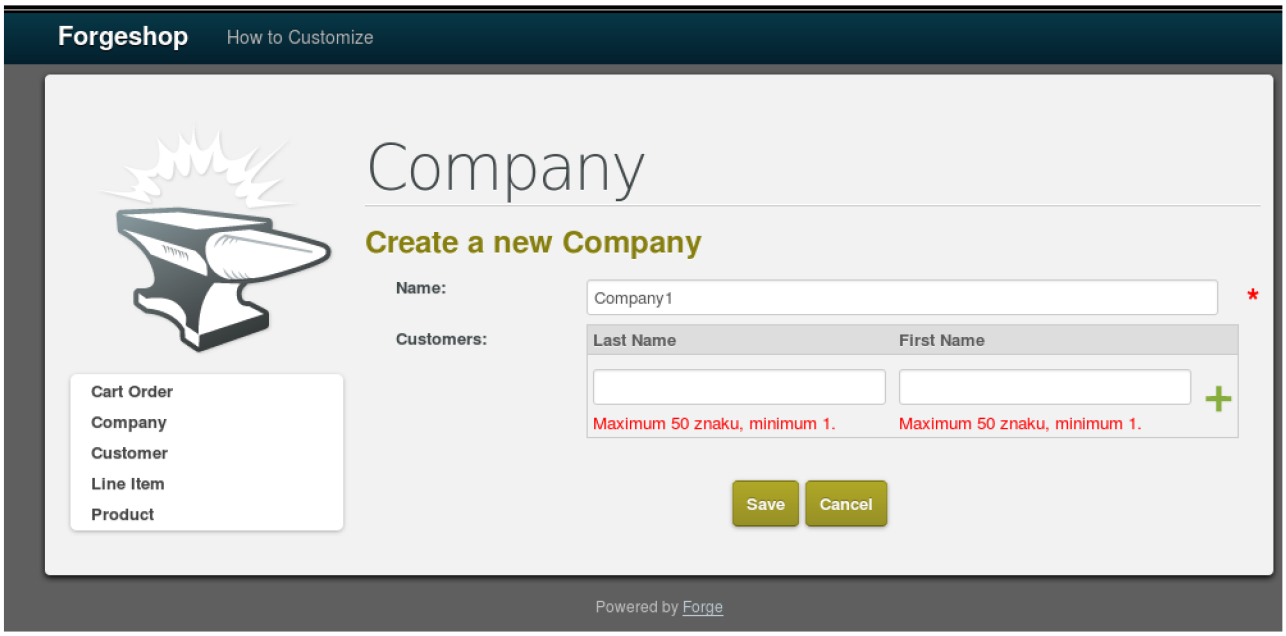
I) Formulář tvorby entity ve vzorové aplikaci: Spring Roo



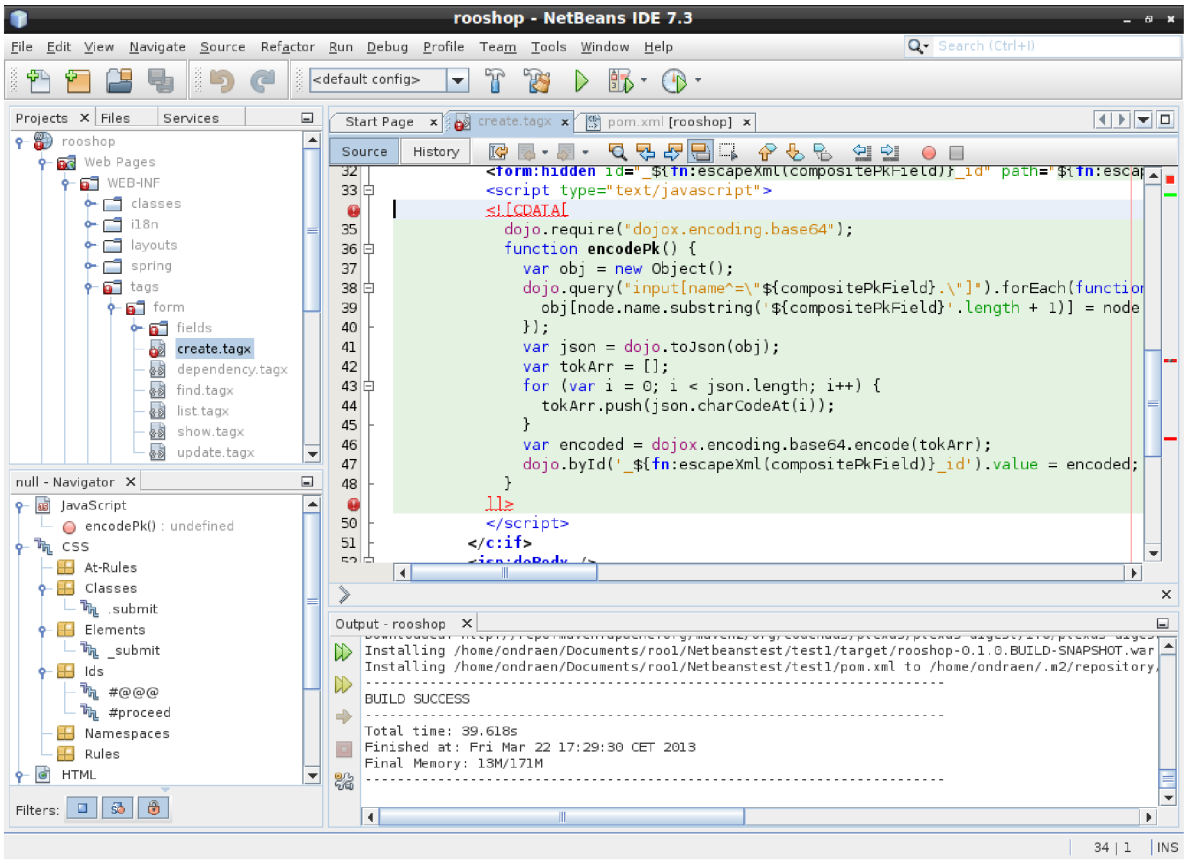
J) Formulář tvorby entity ve vzorové aplikaci: JBoss Forge



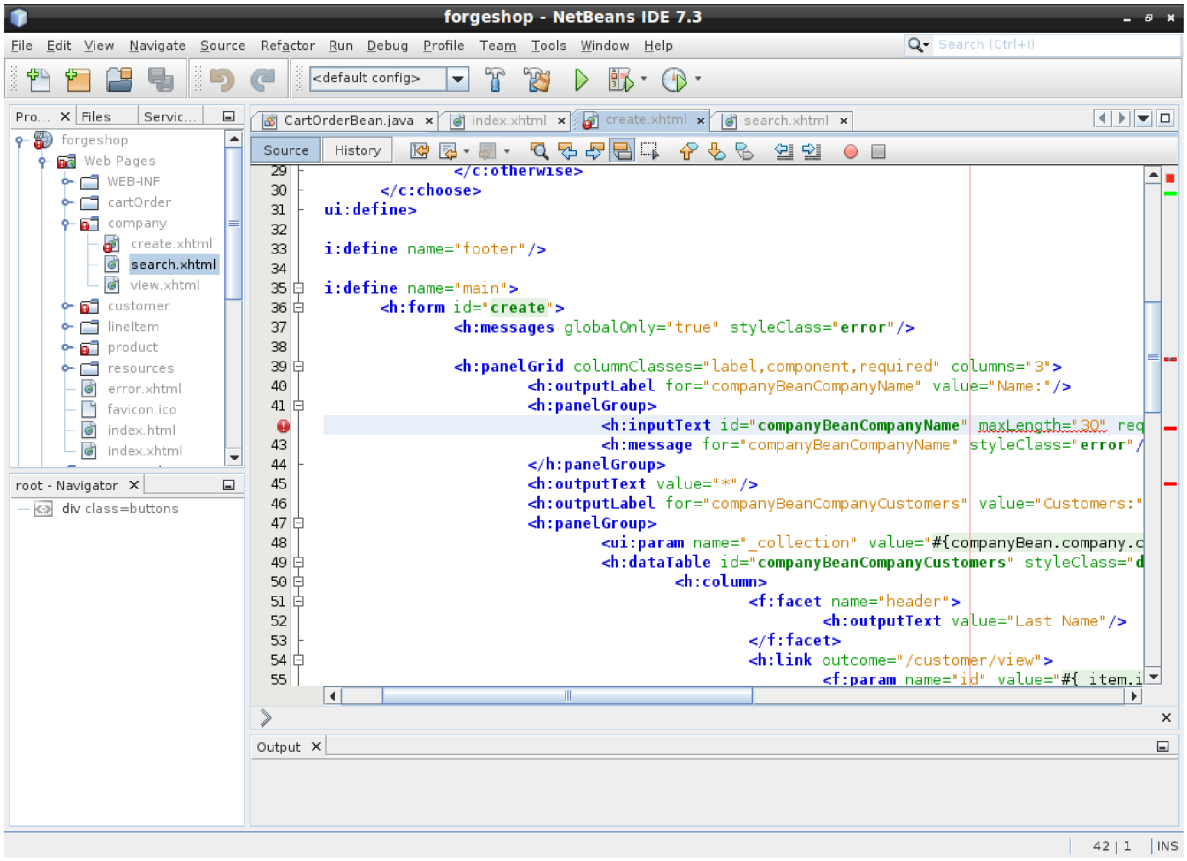
K) Nutnost vyplnit nadbytečná pole ve formuláři vzorové aplikace: JBoss Forge



L) Chyba parsování v prostředí netbeans ve vzorové aplikaci: Spring Roo



M) Chyba v kódu vygenerované aplikace: JBoss Forge



N) Použití Spring Roo v již započatém projektu: nastavení Eclipse STS

```
<buildSpec>
    <buildCommand>
        <name>org.eclipse.wst.common.project.facet.core.builder</name>
        <arguments>
        </arguments>
    </buildCommand>
    <buildCommand>
        <name>org.eclipse.jdt.core.javabuilder</name>
        <arguments>
        </arguments>
    </buildCommand>
    <buildCommand>
        <name>org.springframework.ide.eclipse.core.springbuilder</name>
        <arguments>
        </arguments>
    </buildCommand>
    <buildCommand>
        <name>org.eclipse.m2e.core.maven2Builder</name>
        <arguments>
        </arguments>
    </buildCommand>
    <buildCommand>
        <name>org.hibernate.eclipse.console.hibernateBuilder</name>
        <arguments>
        </arguments>
    </buildCommand>
</buildSpec>
    <natures>
        <nature>com.springsource.sts.roo.core.nature</nature>
        <nature>org.springframework.ide.eclipse.core.springnature</nature>
        <nature>org.eclipse.jdt.core.javanature</nature>
        <nature>org.eclipse.m2e.core.maven2Nature</nature>
        <nature>org.eclipse.wst.common.project.facet.core.nature</nature>
        <nature>org.hibernate.eclipse.console.hibernateNature</nature>
    </natures>
```

O) Obsah přiloženého CD

```
.
├── README
├── nastroje
├── text
│   ├── obrazky
│   └── zdroj
└── vzorove-aplikace
    ├── forgeshop
    └── rooshop
```

- README - soubor popisuje obsah CD a návod instalace nastrojů
- nastroje - Adresář obsahuje archivy ZIP s nástroji Spring Roo a JBoss Forge.
- text - Adresář obsahuje PDF soubor s touto prací. Podsložka zdroj obsahuje zdrojový soubor této práce ve formátu *.tex a *.lyx. Podsložka obrazky obsahuje obrázky použité v této práci.
- vzorove-aplikace - Obsahuje adresáře se zdrojovám kódem vzorových aplikací, včetně sestavených balíčků.