Univerzita Hradec Králové
Fakulta informatiky a managementu

DIPLOMOVÁ PRÁCE

2016 Bc. Tomáš Beran

Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Webový portál pro dodavatele firmy SOMA spol. s r. o.

Diplomová práce

Autor: Bc. Tomáš Beran
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

…..........................

V Hradci Králové dne 1.4.2016 Bc. Tomáš Beran

Poděkování:
Děkuji svému vedoucímu práce doc. Ing. Filipovi Malému, Ph.D. za podnětné rady a připomínky při tvorbě této práce. Děkuji firmě SOMA spol. s r. o. za příležitost zpracovávat webový portál.

Anotace

Cílem této diplomové práce je navrhnout a implementovat webový portál pro dodavatele firmy SOMA spol. s r. o. Společnost v současné době nepoužívá žádný webový portál pro jednotnou komunikaci s dodavateli materiálu nebo služeb. Webový portál rozšiřuje firemní aplikační infrastrukturu o možnost sledovat jednotlivé objednávky ze strany dodavatele bez nutnosti instalace interního systému. Umožňuje také jednotnou komunikaci přímo s interním firemním informačním systémem. Práce je zaměřena na technický popis použitých technologií webového portálu, obsahuje také popis způsobu napojení na stávající informační systém. Závěr práce obsahuje popis samotného nasazování webového portálu do skutečného provozu a popis způsobu bezpečného začlenění do stávající IT infrastruktury s ohledem na dostupnost z internetu a bezpečnost přístupu k interním datům.

Annotation

Title: Web portal for suppliers of SOMA company

The aim of this Diploma Thesis is to propose and implement a web portal for the suppliers to SOMA spol. s r.o. The company currently does not use any web portal for unified communication with materials/services suppliers. The web portal extends corporate application infrastructure by the ability to track individual supplier´s orders without installation of an internal system by the supplier. It also enables a unified communication directly with internal corporate information system. The thesis is focused on a technical description of used web portal technologies, also containing a description of the connection options to the existing information system. The conclusion covers a description of the web portal implementation in real life operation and description of secured integration into existing IT infrastructure based on the Internet access and secured access to internal data.

Obsah

1. Úvod....1
2. Současný stav zpracování objednávek....2
2.1 O firmě SOMA spol. s r. o....2
2.2 Informační systém....2
2.2.1 Popis jednotlivých modulů....3
2.3 Oblast zásobování a kooperace....5
2.3.1 Způsoby uložení a práce s objednávkou....6
2.4 Varianty rozšíření EPR systému....8
3. Popis použitých technologií....10
3.1 Platforma Java 8....10
3.1.1 Garbage collection....11
3.1.2 Novinky Java 8....12
3.2 Apache Tomcat 8....13
3.2.1 Architektura serveru....13
3.3 Spring MVC....14
3.3.1 Model View Controller....15
3.3.2 Front controller....16
3.3.3 Architektura Springu....17
3.4 Spring Security framework....24
3.4.1 Důvody použití Spring Security....24
3.4.2 Součásti frameworku....24
3.4.3 Bezpečnostní konfigurace frameworku....25
3.4.4 Omezení přístupu k metodám....31
3.5 Hibernate....31
3.5.1 Hibernate Query Language....34
3.6 Firebird SQL databáze....35
3.6.1 Multi Generation Architecture....36

3.6.2 Garbage collection..40
3.6.3 Časté výkonnostní problémy s databází..43
3.7 Apache Maven..45
3.7.1 Alternativy Maven..47
4. Popis použitých metod a technického řešení portálu......................................49
4.1 Požadované případy užití portálu..49
4.2 Datový model webového portálu..53
4.3 Zabezpečení aplikace..56
4.4 Mapování zdrojů..59
4.4.1 Stahování dat a tiskových sestav..60
4.5 Přístup k datům..60
4.5.1 Třídy pro přístup k datům..61
5. Nasazení portálu do provozu..64
5.1 Nastavení přístupů do sítě – iptables..64
5.2 Stahování dat z vnitřní sítě – CGI skripty..64
5.2.1 Spolupráce mezi WWW serverem a CGI skriptem............................65
6. Shrnutí výsledků..67
7. Závěr a doporučení..68
8. Seznam použité literatury..69
9. Přílohy..72

1. Úvod

Firma SOMA spol. s r. o. v současné době využívá vlastní podnikový informační systém, který řídí většinu činností ve firmě. Informační systém je vyvíjen přímo v ICT oddělení firmy několika programátory, kteří systém vyvíjí na platformě Netbeans platform v jazyce Java. Informační systém je určen pro spouštění na desktopových stanicích, které jsou umístěné přímo ve firmě. Současný informační systém tedy neumožňuje plnohodnotný běh aplikace mimo firemní prostory a vyžaduje správně nainstalované běhové prostředí jazyka Java. Proto pro rozšíření aplikační infrastruktury firmy byla zvolena cesta webového portálu, který je přístupný téměř z jakéhokoli zařízení, které obsahuje webový prohlížeč a má přístup k internetu.

Webový portál je určen pro dodavatele firmy SOMA, kteří dle objednávek z firmy dodávají materiál, služby nebo zpracovávají technologicky náročné operace. Tito dodavatelé mohou díky webovému portálu sledovat stav objednávek, komunikovat skrz webový portál přímo s firemním informačním systémem a jednotně sledovat požadované změny a urgence termínů. Pracovníci firmy SOMA budou dále využívat pouze interní informační systém. Ten jim však bude umožňovat, díky společné databázi s webovým portálem, sledovat i reagovat na požadavky z portálu, přidělovat přístupové údaje i upravovat zobrazované informace na webovém portálu.

Cílem této práce je navrhnout a implementovat funkční webové rozhraní pro dodavatele firmy SOMA. Práce obsahuje podrobnější popis použitých technologií při vývoji webového portálu. Jedná se technologie použité v back-endové části portálu (MVC webový framework, aplikační server, databáze, ...). Popis jednotlivých technologií je zaměřen na funkce a vlastnosti, které jsou přímo využity při vývoji webového portálu nebo při jeho nasazování do provozu. Práce také obsahuje důležité nebo funkčně zajímavé části zdrojového kódu samotné aplikace webového portálu. V diplomové práci jsou dále popsány problémy při nasazování do skutečného provozu. Tyto problémy vycházejí zejména z firemní bezpečnostní politiky firmy, kdy volně přístupný portál má jen omezené možnosti, jak přistupovat k firemním datům a jak je prezentovat uživatelům portálu.

2. Současný stav zpracování objednávek

Tato část diplomové práce popisuje stav zpracování objednávek před nasazením webového portálu. V úvodní části je krátce představena firma SOMA, které se webový portál týká a pro kterou je určen. V podkapitolách jsou popsány jednotlivé moduly firemního informačního systému, pomocí kterých zaměstnanci pracují a přistupují k datům. Podrobněji je pak popsán mechanismus objednávání a stávající komunikace s dodavateli, dále pak rozdílný způsob ukládání dat o objednávkách kooperace a objednávkách materiálu. V závěru této části jsou popsány dvě uvažované varianty rozšíření informačního systému a důvody pro výběr webové aplikace.

2.1 O firmě SOMA spol. s r. o.

Firma SOMA založena roku 1993 se sídlem v Lanškrouně (okres Ústí nad Orlicí) se zabývá výrobou flexotiskových zařízení. V současné době zaměstnává cca 250 zaměstnanců. Jednotlivá zařízení se vyrábí dle specifikace zákazníka, jedná se tedy o zakázkovou výrobu. Lze říci, že každý vyrobený stroj je unikátní. Firma má vlastní konstrukční oddělení, kde se stroje navrhují dle specifikace každého zákazníka. Jednotlivé stroje se skládají z několika tisíc nakupovaných a vyráběných dílů. Mezi charakteristické vlastnosti těchto flexotiskových strojů patří rozměry okolo 10 x 3 x 3 metry, rychlost tisku až 300 m/minutu a propracovaný design. Zákazníci těchto strojů pochází z celého světa, největší podíl však mají země na východ od České republiky (Rusko a země Asie).

Firma obsahuje vlastní IT oddělení, které se zabývá vývojem interního informačního systému, spravuje firemní hardwarovou infrastrukturu (servery, osobní stanice, tenké klienty, notebooky, telefony apod.) a zajišťuje správný chod dalšího podpůrného softwaru různého zaměření. Hlavní pracovní náplní IT oddělení je vývoj EPR systému (dále také Informační systém nebo IS).

2.2 Informační systém

V současné době je ve firmě vyvíjen vlastní informační systém malým týmem IT pracovníků (zpravidla 2-3 IT pracovníci). Informační systém je vyvíjen v programovacím jazyce Java pomocí Netbeans platformy. K běhu informačního systému třeba mít správně nainstalované běhové prostředí Java. Díky tomu je systém možné spouštět na jednom ze tří nejpoužívanějších operačních systémů (Windows, Linux, OS X). Systém běží na vnitřní firemní síti a není možné ho standardně spouštět mimo vnitřní firemní LAN. Jedinou možností je přihlásit se vzdáleně na vzdálenou plochu jednoho ze tří terminálových serverů, nebo se přihlásit vzdáleně do sítě pomocí VPN. Možnosti vzdálené plochy využívají hlavně zaměstnanci na cestách nebo ti, kteří pracují mimo firmu, VPN připojení používají pouze IT

pracovníci pro vzdálenou správu systémů a uživatelskou podporu. Data informačního systému jsou uložena v databázi Firebird, ke které uživatelé přistupují právě pomocí vnitřního informačního systému.

Informační systém obsahuje důležité součásti pro řízení firmy (jako většina ERP systémů). Mezi hlavní oblasti patří například oblast ekonomická (finance, účetnictví, personální, majetek), obchodní (marketing, servis, management), TPV nebo výroba. Každý takový modul obsahuje několik podoblastí, se kterými uživatelé pracují. Podoblasti jsou interně nazývány agendami, kde každá agenda má svojí specifickou funkci a zaměření.

2.2.1 Popis jednotlivých modulů

- Modul Finance – slouží pro ekonomické oddělení. Zde se evidují bankovní operace nebo pokladní operace, evidují se zde salda závazků i pohledávek a zpracovávají se zde zálohové listy.
- Modul Účetnictví – modul je úzce spjat z předchozím modulem finance. Zde se pracuje s dodavatelskými fakturami, účetními předpisy a dalšími oblastmi spadající pod oddělení účtárny.
- Modul Personalistika – slouží pro evidenci pracovníků, definici jejich hierarchie (nadřízenost podřízenost) a popis jejich činnosti ve firmě.
- Modul Majetek – modul slouží pro práci s firemním majetkem (evidence, vyřazování, apod.).
- Modul Marketing – jedná se o jednu z obsáhlejších oblastí. Zde se pracuje s obchodními případy, evidují se zde firmy a komunikace s nimi.
- Modul Servis – modul slouží pro servisní oddělení. Zde si evidují svoje servisní případy, servisní nabídky nebo reklamační řízení. Zde je také umístěna evidence vyrobených strojů.
- Modul Management – poměrně nový modul, který slouží pro managerské vyhodnocování různých oblastí výroby a zásobování.
- Modul Plán a odbyt – tento modul slouží pro oddělení plánu. Zde existuje hlavní evidence jednotlivých zakázek, oblast odbytu je zde zastoupena agendou odběratelských faktur.
- Modul TPV – modul TPV (Technická příprava výroby) obsahující různé agendy pro práci konstrukčního oddělení. Zde se evidují jednotlivé části prototypů strojů a dle specifikace stroje se zde plní zakázky. Zde jsou také některé agendy určené pro import a integraci dat z jiných systémů.

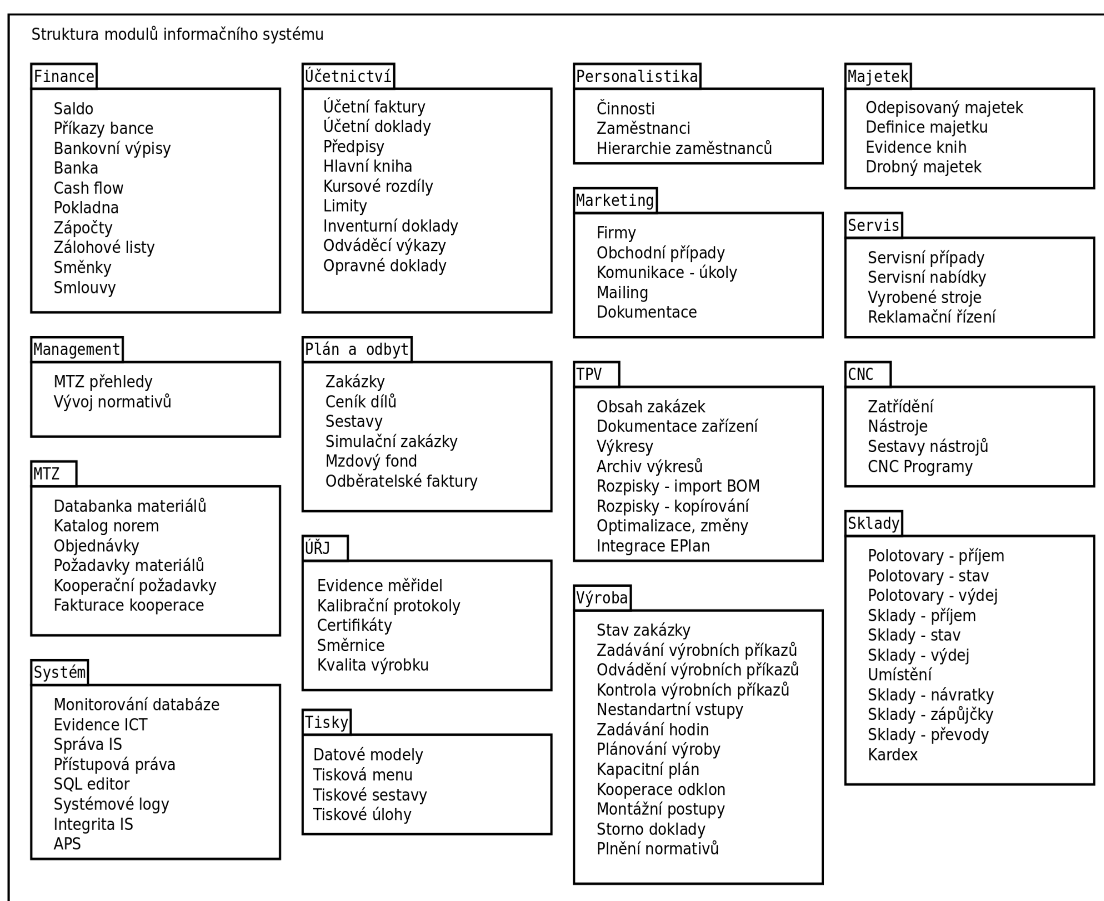
- Modul CNC – modul slouží pro CNC programátory k evidenci jejich programů a součástí programů.
- Modul MTZ – modul MTZ (Materiálně technické zabezpečení) slouží k práci nákupního a kooperačního oddělení. Zde se evidují nakupované díly, požadavky na nakupované díly, kooperační požadavky (výrobní operace, které bude provádět externí firma, například z technologického nebo kapacitního důvodu). Modul obsahuje agendu pro práci s objednávkami a fakturaci objednávek.
- Modul ÚŘJ – modul slouží pro oddělení řízení jakosti, obsahuje agendy pro evidenci tohoto oddělení. Modul obsahuje agendu pro výstupní kontrolu vyrobených strojů, která se na každém vyrobeném zařízení provádí.
- Modul Sklady – modul obsahuje agendy pro práci ve skladech, obsahuje sledování stavu zásob nakupovaných nebo vyráběných dílů. Obsahuje agendy pro příjem nebo výdej dílů na sklad, agendy pro různé převádění dílů mezi sklady nebo také sledování konkrétních umístění ve skladech.
- Modul Tisky – modul je určen pro práci IT oddělení. Zde se vytváří datové modely a nastavují tiskové sestavy napříč celým systémem.
- Modul Systém – modul určen pro evidenci IT oddělení. Obsahuje agendy pro nastavování přístupových práv do IS, sledování IS (logy, připojení, statistiky) a agendy pro další podpůrné nástroje IS a IT oddělení.
- Modul Výroba – nejobsáhlejší modul, co se týče počtu uživatelů a počtu agend. Modul obsahuje agendy pro řízení výroby, sledování stavu jednotlivých zakázek, plánování montáže jednotlivých celků, řízení oddělení obrobny nebo agendu pro odklon dílů do kooperace. Modul také obsahuje agendu pro vyhodnocování plnění normativů.

Na obrázku 1 jsou agendy rozděleny do jednotlivých modulů, většina modulů navíc obsahuje tzv. číselníky, což jsou vlastně jednoduché tabulky definující určitou oblast (například číselník měny, bankovní účty, kódy nákladů, apod.), číselníky nejsou na obrázku 1 zobrazeny.

Interní informační strukturu ještě doplňují některé další (zpravidla externě dodávané) systémy, které jsou s IS synchronizovány. Mezi externě dodávaný systém patří například systém pro pokročilé plánování (Factory Planner), systém pro vývoj konstrukčních výkresů (ProENGINEER, Eplan), systém pro správu konstrukčních dat (PTC Windchill) nebo docházkový systém (Anet Time). Tyto systémy poskytují data, která jsou různými způsoby importována do IS, kde jsou dále používána a zpracovávána. Novinkou vyvíjenou IT oddělením je také Servisní portál, je to webová aplikace přístupná z internetu, která slouží pro zákazníky, kteří si zakoupili stroj a chtějí sledovat různé informace týkající se právě jejich

strojového parku.

Přístup k informačnímu systému povoluje IT oddělení na pokyn vedoucích jednotlivých oddělení. Před používáním systému se musí každý uživatel přihlásit. Přihlášení je pak realizováno na úrovni databáze, tzn. systém se připojí k databázi pod zadanými uživatelskými přístupy a databáze sama drží informace, ze kterých tabulek může uživatel číst, do kterých zapisovat nebo které procedury může spouštět. V systému se také drží informace, do kterých agend může uživatel přistupovat, které tabulky daná agenda potřebuje pro práci a také jakou úroveň používání agendy uživatel má (pouze zobrazení dat, editace dat, …).

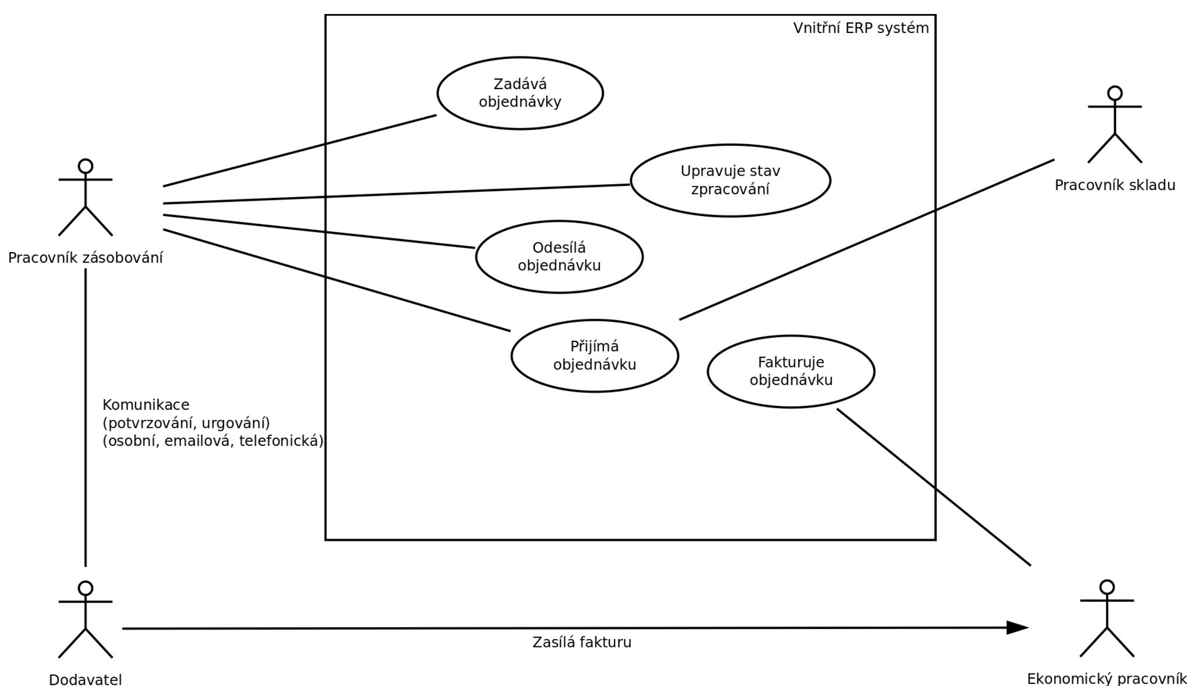


Obrázek 1: Obsah jednotlivých modulů IS. [autor]

2.3 Oblast zásobování a kooperace

Současný informační systém umožňuje evidovat a zpracovávat objednávky. Tato oblast zasahuje v IS do různých částí systému: Objednávky, Sklady, Účetnictví, Fakturace, Výroba a další. Do informačního systému však přistupují pouze zaměstnanci firmy a nikoli dodavatelé. Komunikace mezi IS a dodavateli funguje pouze jednostranně, systém umožňuje odesílat

objednávku dodavateli pomocí emailu, kde v příloze je obsah objednávky a případné doplňkové informace (výkresy objednávaných dílů, CNC data pro zpracování, …). Další komunikace a případné nejasnosti (cenové, specifikační, termínové) jsou realizovány individuálně s pracovníky příslušného oddělení (Nákup, Kooperace) a to telefonickou nebo emailovou komunikací. Komunikace není nikde zaznamenávána (nepočítaje historie jednotlivých emailových schránek) a do IS se pak dostává pouze úprava objednávky, na které se obě strany dohodly. Z obrázku 2 je tedy patrné, že komunikace s dodavateli probíhá výhradně mimo IS, nejednotným způsobem.



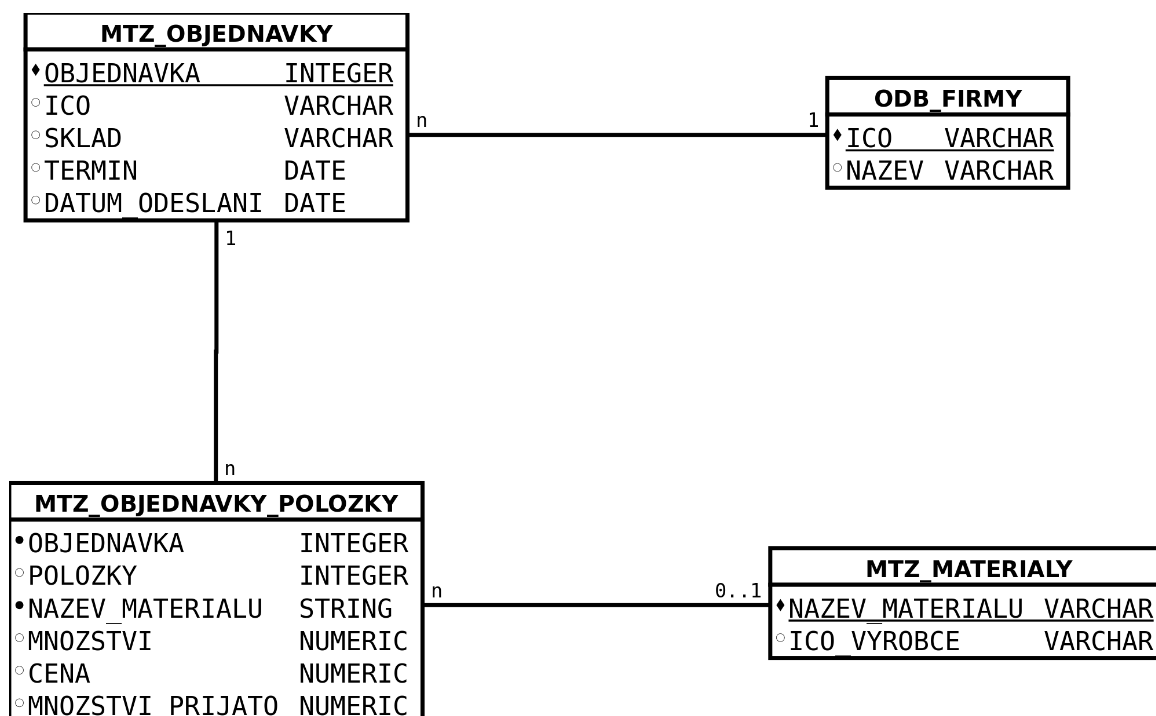
Obrázek 2: Diagram stávající komunikace. [autor]

2.3.1 Způsoby uložení a práce s objednávkou

Systému podporuje dva typy objednávek. Jedná se o objednávky materiálu a objednávky operací/postupů (interně se tento proces nazývá kooperace). Objednávky materiálu se provádí dle aktuálních požadavků z výroby, kdy se objednávají díly, které nejsou v dostatečném množství na skladě a blíží se jejich termín zpracování nebo montáže do stroje. Objednávky kooperací se dělí na kapacitní a technologickou kooperaci, ale z pohledu objednacího cyklu je zpracování kooperací stejné. Technologická kooperace se provádí v případech, kdy firma není sama schopna daný úkon s dílem provést (nejsou k tomu prostředky). Příkladem technologické kooperace je laserové řezání dílu, kdy firma nevlastní stroj, který by toto řezání provedl. Kapacitní kooperace se provádí z důvodu malé kapacity nebo velkého vytížení určitého pracoviště. Objednávka nabývá několika stavů dle vyplněných sloupečků, při vytváření objednávky a plnění položkami je objednávka ve stavu vytváření. Pokud je nastaven

Datum odesláni poptávky, reprezentuje záznam v tabulce MTZ_OBJEDNAVKY poptávku. Pokud dodavatel poptávku akceptuje, pracovník nastaví datum odeslání a záznam se stává objednávkou. Na konci cyklu zpracování objednávky se nastaví datum vyřízení – objednávka je vyřízena.

Rozdíl kooperační a materiálové objednávky je také ve způsobu uložení dat do databáze. Na obrázku 3 jsou zobrazeny tabulky použité pro uložení materiálové objednávky. Tabulka MTZ_OBJEDNAVKY obsahuje hlavičky jednotlivých objednávek, tabulka obsahuje různé informace o objednávce (číslo, IČO firmy, různé termíny a data). Tabulka MTZ_OBJEDNAVKY_POLOZKY obsahuje jednotlivé položky objednávky, položkou může být materiál z tabulky MTZ_MATERIALY nebo třeba služba (úklidové práce), která není v číselníku materiálů. Každá firma vedená v systému je uložena v tabulce ODB_FIRMY, která obsahuje všechny důležité informace o dané firmě.

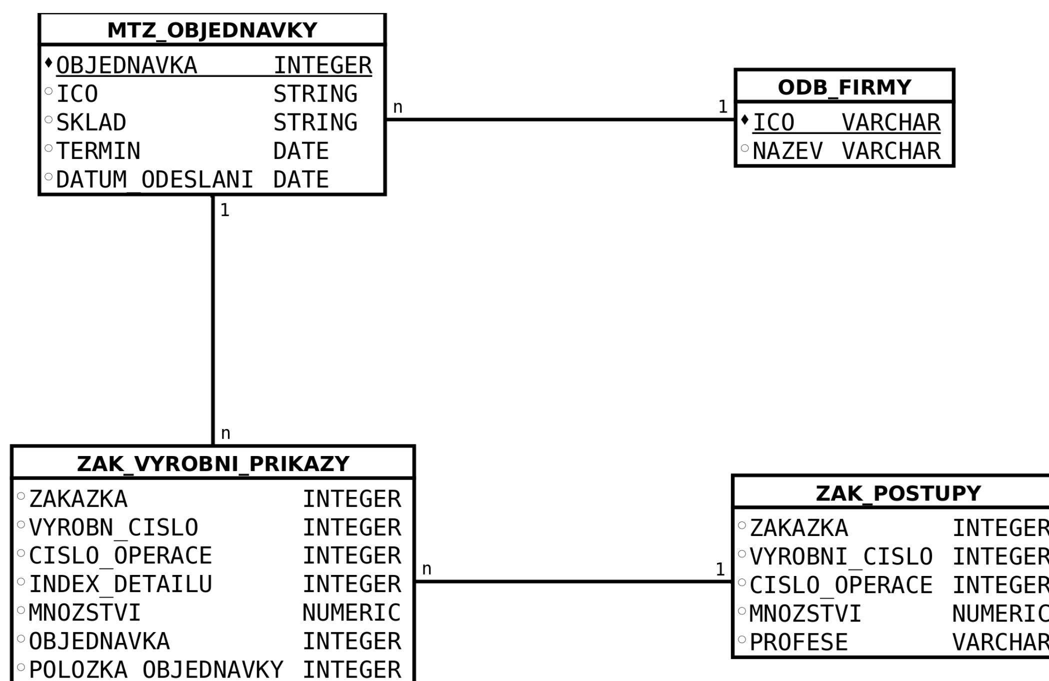


Obrázek 3: Tabulky materiálové objednávky. [autor]

Kooperační objednávky mají položky uložené v jiných tabulkách než materiálové (viz obrázek 4). Hlavička objednávky a firmy zůstává stejná, ale objednávka již neobsahuje položky v tabulce položek, ale v tabulce výrobních příkazů. Každý díl má svůj postup (seznam operací nutných k vyrobení dílu), každá operace (ZAK_POSTUPY) může být zpracována na různých pracovištích (tabulka ZAK_VYROBNI_PRIKAZY). Pokud v tabulce výrobních příkazů je vyplněno číslo objednávky a položka objednávky, tak se jedná o příkaz zpracovávaný v kooperaci. Pokud je díl zpracovávaný celý v kooperaci a obsahuje například

tři výrobní příkazy, tak všechny tři výrobní příkazy budou mít stejné číslo objednávky a stejné číslo položky. Společně tak budou tvořit jednu položku objednávky. O zobrazení jednotlivých položek objednávky se stará procedura, která slučuje výrobní příkazy dle čísla objednávky a čísla položky.

Oba případy se liší nejen z pohledu uložení dat do tabulek, ale také způsobem, kterým se objednávky vyřizují. U materiálové položky se položka přijímá pomocí sloupečku MNOZSTVI_PRIJATO v tabulce MTZ_OBJEDNAVKY_POLOZKY. Tuto akci provádí pracovníci skladu v agendách k tomu určených. U kooperačních objednávek se položka vyřizuje pomocí výrobního příkazu a sloupců: Datum kontroly a Množství přijato. Tuto akci provádí pracovníci kooperace nebo pracovníci dispečinku při příjmu dílu zpět do firmy.



Obrázek 4: Struktura tabulek kooperační objednávky. [autor]

2.4 Varianty rozšíření EPR systému

Vedle již popsaného interního ERP systému, sloužícího výhradně po zaměstnance firmy, je už delší dobu jasné, že je potřeba začít vyvíjet nástroj, který umožní sjednotit komunikaci mezi firmou SOMA a dodavateli této firmy. Nabízí se více možností, jak část ERP systému zpřístupnit také externím firmám. Jednou z možností je přímo rozšířit ERP systém o agendu, která bude sloužit dodavatelům (obdoba dnešní části objednávky), druhou možností je vytvořit webovou aplikaci, která bude zjednodušenou variantou dnešních objednávek. Tabulka 1 zobrazuje klady a zápory obou navržených variant.

	Rozšíření ERP o další agendu	**Vlastní webová aplikace**
Snadný vývoj / rozšíření aplikace	+	-
Snadné zpřístupnění z internetu	-	+
Běhové prostředí	-	+
Hardware na straně klienta	-	+

Tabulka 1: Porovnání dvou navrhovaných variant rozšíření ERP. [autor]

Prvním bodem v tabulce je snadný vývoj, rozšíření stávajícího EPR systému není tak časové náročné jako vývoj nové aplikace, protože veškeré potřebné komponenty a technická řešení jsou již naprogramované, stačí tedy pouze specifikovat jaká data se budou dodavatelům zobrazovat a jaké funkce budou dostupné. Naproti tomu u vývoje webové aplikace je potřeba většinu programátorské práce provést od začátku, je to tedy časově mnohem náročnější.

Následující body spolu více či méně souvisí. Hlavním požadavkem na rozšíření je to, aby uživatel (dodavatel) měl snadný přístup k požadovaným informacím. Dodavateli SOMA nejsou pouze velké firmy, ale jsou mezi nimi i menší podnikatelé či živnostníci, kteří nemusí mít vždy k dispozici zařízení, které by jim umožňovalo přístup do EPR systému na platformě Java. K běhu ERP systému je potřebné mít systém správně nainstalovaný, správně nastavené běhové prostředí Java a také podporovaný hardware a operační systém. Oproti tomu webovou aplikaci lze používat dnes již na většině mobilních telefonů, tabletů, notebooků nebo pevných stanic, a to bez ohledu na použitý operační systém nebo hardwarovou architekturu. Zjednodušeně řečeno, stačí pouze internetový prohlížeč a internetové připojení.

3. Popis použitých technologií

Cesta vývoje vlastního webového portálu je dle předchozích kapitol jasná. Otázkou je, jaké technologie při vývoji použít. Vzhledem k vývoji interního ERP systému na platformě Java (J2SE) a mnohaletým zkušenostem IT pracovníků s vývojem v jazyce Java, jsou jiné programovací jazyky (APS. NET, PHP, C++) prakticky vyloučeny. Volba tedy padla na vývoj JEE aplikace. Jako webový server se již ve firmě využívá Apache Tomcat, běžící na samostatném dedikovaném serveru. Na tomto serveru zatím běží pouze servisní portál, který není zdaleka tak využívaný, jako se předpokládá u portálu pro dodavatele. Typ databáze je také předem určený, protože stávající systém je celý postaven na Firebird databázi a volit pro webový portál jiný typ by nebylo vhodné, například z důvodu převodu a synchronizace dat mezi dvěma různými databázemi.

Pro vývoj webového portálu pro dodavatele (dále také Kooperační portál) je použita aktuální verze jazyka Java (Java 8) a webový server Apache Tomcat 8. Základem celého webového projektu je framework Spring MVC, který v sobě zahrnuje moderní pojetí MVC aplikací. Bezpečnost aplikace řeší další z této rodiny frameworků Spring Security. Jako správce knihoven a automatických buildů je použit systém Maven, který je v použitém IDE prostředí Netbeans IDE 8.0 velice dobře podporován a nevyžaduje žádnou náročnější konfiguraci, protože již v základu obsahuje všechny používané funkce potřebné pro vývoj. V následujících kapitolách jsou tyto technologie podrobněji popsány.

3.1 Platforma Java 8

Java 8 je revoluční verze nejpoužívanější světové platformy pro vývoj aplikací. Dle nejpoužívanějšího komunitního portálu pro programátory stackoverflow.com (stackoverflow.com/research/developer-survey-2015#tech-lang) je první mezi programovacími jazyky (před Java je pouze javascript a SQL). Tato verze zahrnuje mnoho vylepšení standardního modelu programování v jazyce Java, řídí evoluci JVM a standardních knihoven tohoto jazyka. Java 8 obsahuje mnoho nových funkcí pro produktivní, snadné i vícejazyčné programování (lokalizované aplikace), bezpečnost a výkonnost. Java 8 je zatím poslední verze největší, otevřené, standardizované a komunitně založené platformy. [ORACLE]

Java je objektově orientovaný programovací jazyk, v současnosti vlastněný společností Oracle. Java je rozdělena do dvou hlavních distribucí: JRE a JDK. JRE (Java Runtime Enviroment) obsahuje součásti nutné pro běh Java SE aplikací, prostředí určeno zpravidla pro koncové uživatele. Druhou částí je JDK (Java Development Kit). Jedná se o balík určený hlavně pro vývojáře softwaru, protože obsahuje nástroje potřebné pro vývoj aplikací v Java jazyce. Java obsahuje prostředek pro automatickou správu paměti nazývanou Garbage Collection.

3.1.1 Garbage collection

Pro psaní kvalitního kódu je samozřejmě nutné znát základní syntaxi jazyka, nicméně alespoň základní znalost, jak funguje správa paměti, může pomoci psát mnohem lepší a rychlejší Java aplikace. Garbage collection (dále také GC) se stará o správu paměti, díky tomu není nutné paměť uvolňovat manuálně jako v jiných programovacích jazycích (např. C). GC paměť uvolňuje pokud je rezervována pro objekt, na který již neexistuje žádná reference (jednoduchým příkladem může být vnitřní atribut metody, která již skončila). Tento algoritmus je někdy nazýván také jako Algoritmus počítání referencí.

GC rozděluje paměť pro program do tří hlavních částí: Young generation, Old generation a Permanent generation. První část Young generation slouží jako paměť pro nové objekty, GC zde provádí kontrolu nepotřebných objektů nejčastěji. Díky časté kontrole se nepotřebné objekty z paměti uvolní rychleji a nezabírají paměť dlouho. Young generation je rozdělena do tří částí: 1x eden space a 2x survivor space. Většina nových objektů je uložena v Eden space, pokud objekt přežije jeden cyklus kontroly referencí, je přesunut do jedné z částí survival space.

Pokud objekt přežije několik cyklů kontroly GC v první části, překopíruje se do druhé části nazývané Old generation. Zde jsou objekty s delší životností, není tedy nutné provádět kontrolu tak často jako v Young generation. Tato část paměti je také větší než paměť alokovaná pro Young generation. Poslední část paměti nazývaná Permanent generation slouží pro ukládání metadat o třídách a class-loaderech, obsahuje bytecode jednotlivých metod a také interní objekty vytvořené JVM. Více se lze dočíst v [CUBRID].

	Total: -Xms				Total: XX: MaxPermSize	
Virtual or reserved	Eden	Survivor	Tenured	Virtual or reserved	PermGen	Virtual or reserved
Young Generation -(-XX:MaxNewSize)			Old Generation		Permanent Generation (-XX: MaxPermSize)	
Heap Memory (-Xmx)					Non-Heap	
JVM Total Memory Distribution						

Obrázek 5: Rozdělení paměti JVM. [JAVAHONK]

Na obrázku 5 je grafické znázornění rozdělení paměti do oblastí v JVM. Jsou zde také naznačeny parametry, které lze použít pro zvýšení dané části paměti (např. -Xmx2048m nastaví maximální velikost Heap paměti na 2048 MB). Při spouštění aplikace pomocí příkazu java -jar, můžeme tyto parametry přidat a tím zvýšit nebo naopak snížit některou z paměťových oblastí.

3.1.2 Novinky Java 8

Novinek v poslední verzi Java 8 je hned několik. Zajímavou novinkou, ovlivňující přímo styl psaní zdrojového kódu je tzv. Lambda expression. Zjednodušeně řečeno, takto novinka umožňuje psát anonymní metody bez deklarace, bez modifikátoru přístupu (private, public, …), bez návratových hodnot nebo názvu metody. Výhodou je velice krátký zápis a možnost deklarace těla metody v místě volání. Z bezpečnostních novinek je nutné zmínit výchozí povolení TLS 1.2, silnější algoritmy pro šifrování pomocí hesel, nový typ Domain KeyStore java.security. DomainLoadStoreParameter. Z novinek JavaFX je určitě zajímavé nové zobrazovací téma Modena (fxexperience.com). Byla také implementována podpora použití Swing v JavaFX aplikacích. Mezi nově přidané komponenty patří DatePicker nebo TreeTableView. Hodně novinek je také v oblasti 3D grafických funkcí. Novinkou je také dostupnost JavaFX na platformách ARM. Více novinek lze dohladat na [ORACLE].

Následuje ukázka zdrojového kódu bez použití Lambda expression a s použitím novinky Lambda expression (zdroj oracle.com).

```
public static void main(String... args) {
	Runnable r = new Runnable() {
		public void run() {
			System.out.println("Hello world!");
		}
	};
	r.run();
}
```

S použitím Lamda expression Java 8 lze zdrojový kód této metody zkrátit následovně:

```
public static void main(String... args) {
	Runnable r = () -> System.out.println("Hello world!");
	r.run();
}
```

3.2 Apache Tomcat 8

Apache Tomcat je open-source webový server a servlet kontejner vyvíjený pod licencí Apache Licence version 2. Oproti svým konkurentům se jedná o jednodušší a méně robustní implementaci. Apache Tomcat slouží pro spouštění Java Servletů a JavaServer Pages (JSP) aplikací a Java Expression Language (EL). Apache Tomcat je velice stabilní server, který obsahuje všechny funkce, jaké poskytují komerční webové kontejnery a je tak vhodný i pro větší webové projekty. [VUKOTIC]

Aktuální verze serveru je Apache Tomcat 9, která ale není zatím ve stabilní verzi (označenou jako Stable), proto je zde popsána a při vývoji použita poslední stabilní verze Apache Tomcat 8. V tabulce 2 jsou zobrazeny hlavní verze webového serveru spolu s podporovanými verzemi jednotlivých technologií (stav k prosinci 2015).

Apache Tomcat	Servlet	JSP	EL	Podporovaná verze JDK
9.0 (alpha)	4.0	2.4	3.1	1.8
8.0	3.1	2.3	3.0	1.7
7.0	3.0	2.2	2.2	1.6
6.0	2.5	2.1	2.1	1.5

Tabulka 2: Tabulka několika posledních verzí Apache Tomcat serveru s verzemi podporovaných technologií. [TOMCAT]

Pro základní ovládání webového serveru slouží aplikace Tomcat Manager. K ní přistupujeme přes webové rozhraní, zpravidla na portu 8080. Mezi hlavní funkce této aplikace patří instalace, start, stop nebo odinstalace webových aplikací. Pro přístup do této aplikace se musí použít účet, který má nastavenou roli manager-gui.

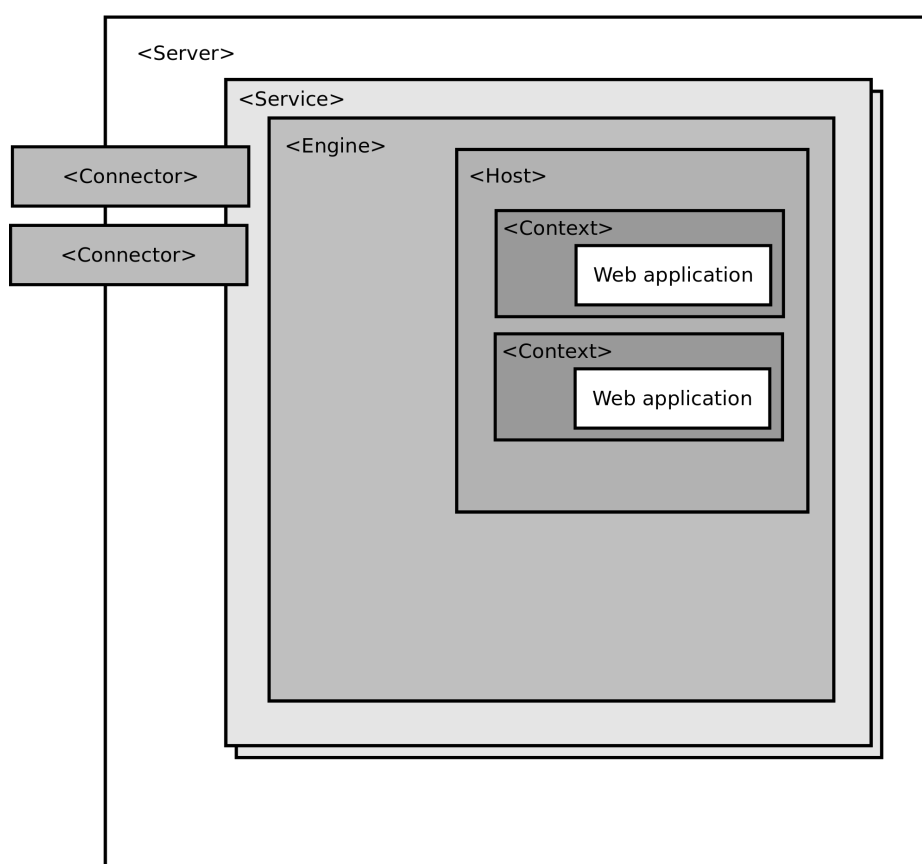
3.2.1 Architektura serveru

Instance Apache Tomcat serveru běží na nejvyšší úrovni hierarchie webového kontejneru. Zpravidla na jednom JVM může běžet pouze jedna tato instance a ostatní Java aplikace běží pod touto instancí webového serveru. Tomcat server je rozdělen do několika úrovní aplikačních kontejnerů dle předem definované hierarchie. Na obrázku 6 je naznačena struktura webového serveru.

Na nejvyšší úrovni hierarchie je <Server>, klíčovou komponentou v této vrstvě je Catalina servlet engine, tato vrstva je na webovém serveru zpravidla v jedné instanci a může obsahovat jednu nebo více <Service> kontejnerů. <Service> element je další úrovní, v této vrstvě se udržuje jeden nebo více <Connector> elementů, kde každý connector sdílí právě jeden <Engine> element. <Connector> elementy se starají o request a response volání jednotlivých

klientských aplikací.

Třetí skupinou v hierarchií je <Engine> element, tento samostatný element v rámci <Service> se stará o všechny požadavky od <Connector> komponent. Dalším elementem v hierarchií je <Host> element, který definuje virtuální hosty, které jsou obsaženy v každé jednotlivé instanci Catalina engine. Každý <Host> může obsahovat (být rodičem) více webových aplikací sdružených pod <Context> vrstvu. Každý <Context> element reprezentuje jednu individuální webovou aplikaci. V rámci <Host> elementu neexistuje žádný limit na počet webových aplikací (context elementů). Informace o hierarchií serveru jsou čerpány z [VUKOTIC].



Obrázek 6: Hierarchie Apache Tomcat serveru s hlavními komponentami.[VUKOTIC]

3.3 Spring MVC

Ačkoli Java Servlet API samo poskytuje velice bohatý a uživatelsky nastavitelný framework pro vývoj webových aplikací, v profesionálním vývoji aplikací ani to nemusí vždy stačit. Tato kapitola se bude zabývat webovým frameworkem Spring MVC, který je velice oblíbeným ze skupiny frameworků Spring jak pro začínající, tak pro pokročilé vývojáře webových aplikací v jazyce Java. Spring MVC je framework založený na návrhovém vzoru Model View

Controller a Front Controller, určený pro vývoj webových aplikací založených na servletech. [VUKOTIC]

Spring MVC framework je patří do skupiny tzv. lightweight frameworků, jedná se o open-source Java framework navržený pro vývoj Java enterprise aplikací. Klíčovou částí frameworku je dependency injection container (dále také DI), který je implementací návrhového vzoru Inversion of Control. DI je navržen pro snažší návrh komplexních aplikací pomocí jednotlivých komponent. Projekt se tak může rozdělit do jednotlivých samostatných částí, které se seskupí právě pomocí DI, tyto části se také nazývají beany a jsou konfigurovány pomocí XML nebo Java anotace. O vytváření instancí a správu těchto komponent se pak stará Spring kontejner (Spring application context), který pak také jednotlivé instance vkládá (injectuje) do jiných komponent. Kromě popsaného DI, Spring také poskytuje mnoho dalších užitečných funkcionalit, mimo jiné přístup k databázi, Aspect oriented programming (AOP), Java Messaging Service (JMS) a mnoho dalších.

V předchozí části kapitoly o Spring MVC frameworku bylo zmíněno, že framework je založen na MVC návrhového vzoru a Front controller. V následujících kapitolách budou zmíněné vzory podrobněji popsány.

3.3.1 Model View Controller

Model View Controller je velice oblíbeným návrhovým vzorem. Základní myšlenkou MVC architektury je oddělení logiky od výstupu. Řeší tedy problém tzv. „špagetového kódu", kdy máme v jednom souboru (třídě) logické operace a zároveň renderování výstupu. Třída tedy obsahuje databázové dotazy, logiku a různé zobrazovací tagy (HTML) nebo komponenty (např. Java Swing komponenty). Takový kód se samozřejmě špatně udržuje a je i špatně čitelný. [ITNETWORK]

Celá aplikace je rozdělena do tří hlavních komponent: Model, View a Controller. Na obrázku 7 je graficky zobrazen možný tok dat mezi jednotlivými komponentami.

Model

Komponenty patřící do části Model obsahují hlavní logiku celé aplikace. Do logiky aplikace patří datový model, databázové dotazy nebo validace dat. Funkce této části spočívá ve zpracování vstupních dat (parametrů) na výstupní data, aniž by věděla odkud data přišla nebo v jakém formátu budou zobrazena. Při použití ORM frameworku Model přímo souvisí s databázovými tabulkami.

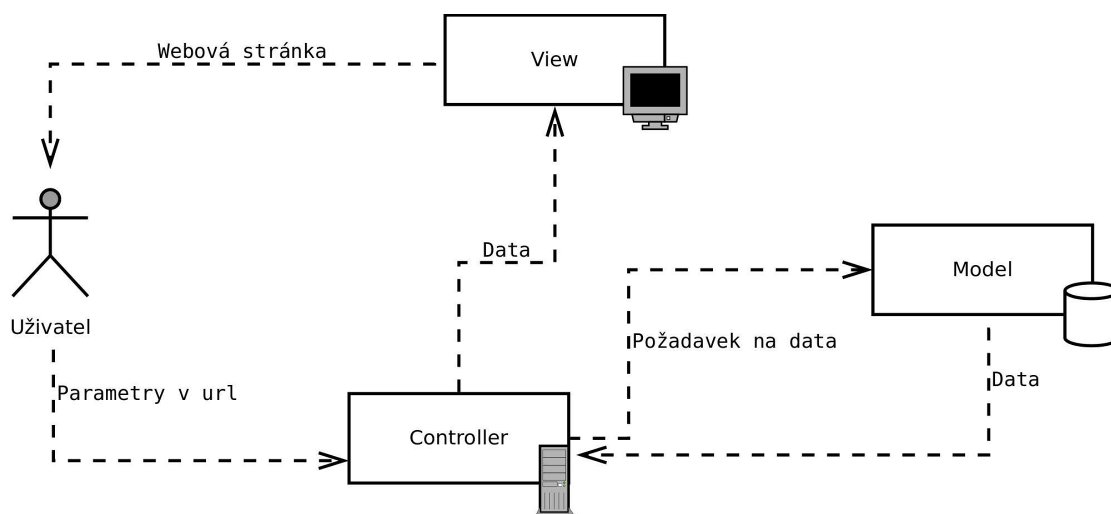
View

Tato část se zabývá samotným zobrazováním dat uživateli. View komponenta zpracovává data z části Model do zvolené vizuální podoby, obvykle by neměla tato část obsahovat žádnou

logiku. Příkladem view technologií webové části aplikace může být například HTML, CSS, JavaScript nebo Java Servlet Pages (JSP) pokud jsou použity s MVC principy (tzn. neobsahují žádnou bussiness logiku a jsou použity pouze pro zobrazení dat).

Controller

Komponenty v části Controller jsou zodpovědné za zpracování a reagování na uživatelské akce. Jedná se chybějící prvek mezi Model a View, který tyto části propojuje a stará se o distribuci potřebných dat mezi komponentami.



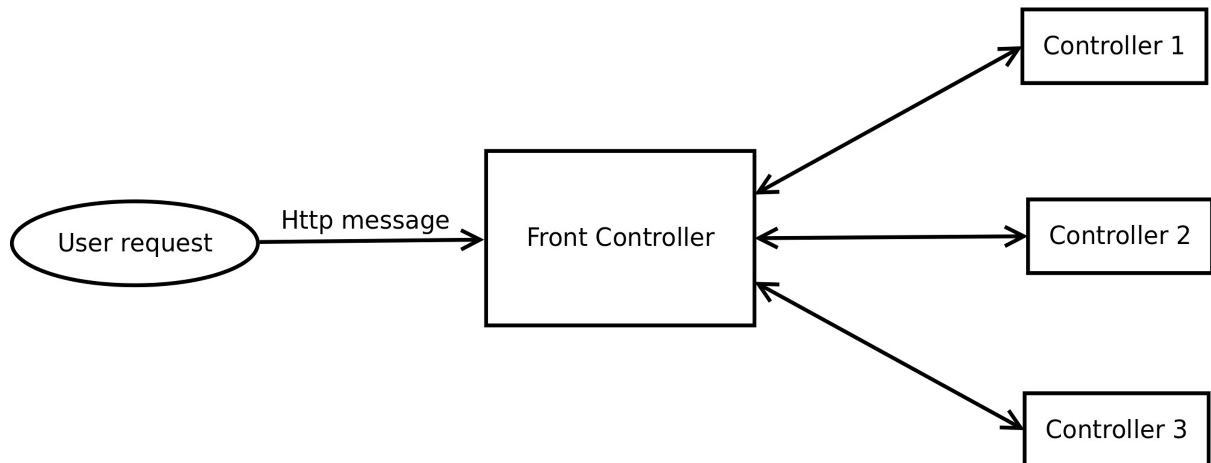
Obrázek 7: Návrhový vzor Model-View-Controller s ukázkou toku dat. [ITNETWORK]

3.3.2 Front controller

Principem front kontroleru je přerozdělovat požadavky (request) uživatele (nebo webové aplikace) na jednom centrálním místě. Role tohoto centrálního bodu (nazývaného Front controller) spočívá v přeposílání jednotlivých požadavků na konkrétní webovou komponentu (specifický kontroler pro určité funkce) a odesílání odpovědi (response) z kontrolérů zpět uživateli. Mimo přeposílání požadavků a odpovědí je Front controller zodpovědný také za manipulaci se session, cache a za celkové filtrování požadavků (například ve smyslu odfiltrování nežádoucích požadavků). [VUKOTIC]

Na obrázku 8 je graficky znázorněn tok požadavku přes front controller.

Z pohledu Spring MVC frameworku je centrální bod implementován třídou DispatcherServlet (z balíku org.springframework.web.servlet.DispatcherServlet).



Obrázek 8: Front controller. [SUN]

3.3.3 Architektura Springu

Při psaní této kapitoly o architektuře Springu bylo čerpáno z knihy Spring in Action, kapitola 1.1 [WINCH].

Spring byl vytvořen s cílem řešit komplexní vývoj enterprise aplikací s použitím JavaBeans. Ačkoli v kontextu Springu se používá slovo beans, neznamená to, že se striktně dodržuje JavaBeans specifikace, komponentou Springu může být jakýkoli objekt POJO. Spring byl navržen hlavně proto, aby zjednodušil vývoj Java enterprise aplikací. Zjednodušení vývoje lze odvodit pomocí následujících čtyř hlavních strategií Springu:

- jednoduchý a neinvazivní způsob vývoje s POJO,
- zrušení pevných vazeb pomocí DI a programování na rozhraní,
- deklarativní programování pomocí aspektů a zavedených konvencí,
- redukce psaní často používaného kódu pomocí aspektů a vzorů.

Téměř u všech vlastností Springu lze vysledovat jednu nebo více ze zmíněných strategií. V následujících kapitolách jsou tyto strategie podrobněji popsány.

Neinvazivní způsob vývoje s POJO

Pod jednoduchým a neinvazivním způsobem vývoje je myšleno, že framework striktně nenařizuje programátorovi, jak má s frameworkem zacházet, jaké komponenty nebo metody používat, nebo z jakých tříd nebo rozhraní je nutné dědit nebo implementovat. Příkladem invazivního programování byly počáteční verze frameworků typu Struts, WebWork nebo třeba Tapestry. Aplikace založená na Springu často obsahuje třídy, které nemusí být nijak speciálně označeny k čemu slouží, nemusí dědit z nějakých speciálních předků. Mohou být třeba pouze

označeny anotací; pak mluvíme o tzv. POJO objektech. Jak již bylo zmíněno výše, POJO (Plain Old Java Objects) existovali již dříve, ale jejich role se výrazně zvýšila s příchodem Spring frameworku a obecně s příchodem Java EE.

Dependency injection

Druhá strategie se zmiňuje o zrušení pevných vazeb mezi třídami a dependency-injection metodě. Jedná se vlastně také o návrhový vzor, kdy při vývoji aplikace (s více třídami navzájem na sobě závislými) nevytváříme instance potřebných tříd přímo v místě potřeby. Zodpovědnost za vytvoření instance přeneseme na DI metodu, která instance vytvoří za nás. Programátor pak pracuje přímo s daným objektem a nestará se o instancování tohoto objektu. Tato metoda zvyšuje čitelnost kódu, umožňuje snadnější testování kódu a usnadňuje pochopení zdrojového kódu. Následuje ukázka kódu bez použití DI a vytvoření instance třídy Trida2 přímo v konstruktoru.

```
public class NejakaTrida implements RozhraniTridy {
  private RozhraniTridy2 trida2;
  public NejakaTrida() {
      this.trida2 = new Trida2();
  } ...
  private void pouzitTridu2() {
      trida2.doSomething();
  }…
```

Jak je z kódu patrné, v konstruktoru třídy NejakaTrida se přímo vytvoří instance třídy Trida2 a to i přesto, že není zatím jasné, zda se instance vůbec použije (tzn. zavolá metoda pouzitTridu2). Ukázka kódu s použitím DI v následující části ukazuje, jak lze také ke stejnému příkladu přistupovat elegantnějším způsobem.

```
public class NejakaTrida {
  @Autowired
  private RozhraniTridy2 trida2;
  public NejakaTrida() {
  } ...
  private void pouzitTridu2() {
      trida2.doSomething();
  }…
```

Ze zdrojového kódu není úplně jasné, kdy se instance třídy Trida2 vytvoří, nicméně pomocí anotace @Autowired Springu programátor sděluje, že právě tuto třídu potřebujeme nainstancovat pomocí DI. Dependency-injection (dle nastavení) může vytvořit instanci tohoto

objektu hned při vytvoření třídy NejakaTrida nebo až při zavolání metody pouzitTridu2. Ke správnému nainjektování je ale nutné, aby někde existovala definice toho, jaká instance se bude vkládat (že se bude jednat o instanci třídy Trida2 a ne jiné, která také může implementovat RozhraniTridy2). Tato definice se může zapsat do konfiguračního souboru s aplikačním kontextem, např. v ApplicationContext.xml ve formátu:

```
<bean id="RozhraniTridy2" class="model.impl.Trida2" />
```

Hlavní přínos takto zapsaného kódu je, že instance tříd mezi sebou nejsou pevně svázány a jejich vazba se definuje zvlášť. Můžeme pak snadno třídu Trida2 vyměnit například za třídu TestovaciTrida2, která také implementuje RozhraniTridy2, ale je vhodnější pro testování nebo jinou situaci vyžadující nahrazení konkrétní implementace. Toto je tedy klíčová výhoda DI – ztráta pevné vazby. Místo použití anotace lze také objekt předat v konstruktoru třídy NejakaTrida - v XML souboru je pak nutné nadefinovat přímo instanci NejakaTrida s konkretní instancí RozhraniTridy2 v konstruktoru. Místo použití XML souboru lze také vytvořit konfigurační třídu označenou anotací @Configuration, ve které stejně jako v XML definujeme konkrétní instance.

Aspektově orientované programování

Třetí zmíněná strategie se zabývá aplikováním aspektů – Aspect oriented programming (AOP).

Aspekt lze chápat jako společnou část, vlastnost nebo potřebu některých metod. AOP zajišťuje ulehčení práce programátora v případech, kdy se při volání metod různých tříd volá vždy stejný nebo podobný kód. Typickým příkladem je otevírání a uzavírání transakce, kontrola práv k provedení operace (autorizace), trasování běhu aplikace atd. Ve Springu lze aspekty využít pomocí anotací (např. @Before, @Aspect, @Pointcut, ...) nebo konfigurace XML souboru. [ZIZKA]

Hlavní výhodu aspektů si lze představit v situaci, kde více tříd obsahuje stejnou část kódu (například logování) a programátor je nucen tuto část kódu pozměnit. Je tedy nutné pozměnit zdrojový kód na více místech současně. Při použití aspektů se zdrojový kód nachází pouze na jednom místě a duplikuje se automaticky do patřičných tříd až při běhu aplikace. Aspekty také mohou celý kód zjednodušit. Příkladem může být otevírání a uzavírání transakcí, které se provádí stále stejně (stejný zdrojový kód). Tuto rutinu tak můžeme z původních metod vyjmout a nahradit pomocí aspektů. Definujeme tedy otevření transakce před voláním určité skupiny metod a uzavření transakce po volání této skupiny metod. Opět se jedná o výhodu, která může výrazně zjednodušit čitelnost a usnadnit porozumění zdrojovému kódu.

Je důležité také zmínit, že aspektové volání metod funguje pouze na instance tříd vytvořené Springem – například definované v XML souboru. Následuje ukázka XML definice třídy NejakaTrida s parametrem v konstruktoru Trida2 spolu s nastavením aspektu na metodu

pouzitTridu2(), kdy před voláním této metody se zavolá metoda z třídy TransactionUtils.open().

```
…
xmlns:aop="‟http://www.springframework.org/schema/aop‟
…
<bean id="NejakaTrida" class="‟model.impl.NejakaTrida">
      <constructor-arg ref="trida2" />
</bean>
<bean id="trida2" class="model.impl.Trida2" />
<aop:config>
      <aop:aspect ref="trans">
      <aop:pointcut id="volaniMetody"
                  expression="execution(* *.pouzitTridu2(..))"/>
            <aop:before pointcut-ref="asp-ref" method="open"/>
      </aop:aspect>
</aop:config>
<bean id="‟volaniMetody‟ class="‟utils.TrasactionUtils‟>
```

Znovu opakující se kód – boilerplate

Při častějším programování podobných projektů, nebo vývoji jedné aplikace, se vývojáři často setkávají s pocitem, že právě psaný kód již někdy psali. Tyto části zdrojových kódu se někdy také nazývají „boilerplate code“. Běžným příkladem boilerplate kódu může být práce s JDBC a SQL dotazem na data do databáze. Následuje příklad získání informací o zaměstnanci z databáze přes ID zaměstnance (příklad převzatý z Spring in Action str. 16 [WALLS]):

```
public Employee getEmployeeById(long id) {
Connection conn = null;
PreparedStatement stmt = null;
ResultSet rs = null;
try {
      conn = dataSource.getConnection();
      stmt = conn.prepareStatement(
                  "select id, firstname, lastname, salary from " +
                  "employee where id=?");
      stmt.setLong(1, id);
      rs = stmt.executeQuery();
      Employee employee = null;
```

```
        if (rs.next()) {
                employee = new Employee();
                employee.setId(rs.getLong("id"));
                employee.setFirstName(rs.getString("firstname"));
                employee.setLastName(rs.getString("lastname"));
                employee.setSalary(rs.getBigDecimal("salary"));
        }
        return employee;
} catch (SQLException e) {
//do something
} finally {
        if(rs != null) {
                try {
                        rs.close();
                } catch(SQLException e) {}
        }
        if(stmt != null) {
                try {
                        stmt.close();
                } catch(SQLException e) {}
        }
        if(conn != null) {
                try {
                        conn.close();
                } catch(SQLException e) {}
        }
}
return null;
}
```

Z ukázky zdrojového kódu je jasně vidět, že pouze malá část kódu specifikuje konkrétní požadavek na údaje o zaměstnanci. Zbytek kódu je nutná obsluha kolem dotazu, která se pro podobné dotazy stále opakuje a je stejná (definice spojení, vytvoření spojení, odchytávání SQL výjimek bezpečné uzavírání spojení, dotazu i result setu). Spring umožňuje tento kód zapouzdřit do tzv. template (vzoru). Například použitím SimpleJdbcTemplate můžeme metodu getEmployeeById přepsat tak, aby obsahovala menší množství zdrojového kódu a

zlepšila se tak čitelnost důležité části s dotazem na databázi. Následuje příklad takto upraveného kódu (příklad převzatý z [WALLS] str. 17):

```
public Employee getEmployeeById(long id) {
return jdbcTemplate.queryForObject(
      "select id, firstname, lastname, salary " +
      "from employee where id=?", new RowMapper<Employee>() {
            public Employee mapRow(ResultSet rs, int rowNum)
                  throws SQLException {
                  Employee employee = new Employee();
                  employee.setId(rs.getLong("id"));
                  employee.setFirstName(rs.getString("firstname"));
                  employee.setLastName(rs.getString("lastname"));
                  employee.setSalary(rs.getBigDecimal("salary"));
                  return employee;
                  }
            },
      id);
```

V přepsané metodě getEmployeeById jsou jasně patrné důležité části kódu, jedná se hlavně o konkrétní dotaz SQL dotaz na databázi„select id, firstname, lastname … from employee where id = ?“, mapování výsledných dat na třídu Employee „employee.setId(rs.getLong(„id“) ...“ a vstupující parametr s identifikací uživatele „id“. Implementaci metody queryForObject můžeme najít ve zdrojových kódech Spring frameworku ve třídě org.springframework.jdbc.core.JdbcTemplate.

```
public class JdbcTemplate extends JdbcAccessor implements
JdbcOperations {
...
public Object queryForObject (String sql, RowMapper rowMapper) throws …
{
      List results = query(sql, rowMapper);
      return DataAccessUtils.requiredSingleResult(results);
}
```

V metodě se vlastně pouze vytvoří instance List a zavolá metoda requiredSingleResult ze třídy org.springframework.dao.support.DataAccessUtils.

```
public abstract class DataAccessUtils {
...
public static Object requiredSingleResult(Collection results)
	throws IncorrectResultSizeDataAccessException {
	int size = (results != null? results.size() : 0);
	if (size == 0) {
		throw new EmptyResultDataAccessException(1);
	}
	if (results.size() > 1) {
		throw new IncorrectResultSizeDataAccessException(1, size);
	}
	return results.iterator().next();
}
```

Obecně lze tedy kód redukovat tak, že obslužný opakující se kód přepíšeme do nějaké metody (v příkladu je použita metoda jdbcTemplate.queryForObject, kde se o zdrojový kód stará samotný framework), která obsahuje parametry, pomocí nichž je schopna znovu opakující se činnost vykonat stejně jako kdyby byl kód psán celý znovu. Tyto metody nemusí nutně pocházet pouze z použitých frameworků, ale je také možné si je napsat přímo. Jako jednodušší příklad lze uvést převod textu na číslo. Pro správný převod musíme ošetřit, aby textová reprezentace nebyla null a současně že se jedná skutečně o číselnou hodnotu vyjádřenou pomocí Stringu. Je zbytečné vždy, když potřebujeme převést text na číslo, psát kód ručně. Stačí si vytvořit například metodu strToIntDef, která text převede a v případě neúspěchu vrátí výchozí zadanou hodnotu.

```
public static Integer strToIntDef(String value, Integer defaultValue) {
	Integer retVal = defaultValue;
	if (value != null) {
		try {
			retVal = Integer.valueOf(value);
		} catch (NumberFormatException ex) {
		}
	}
	return retVal;
}
```

Z ukázky je zřejmé, že převod je skutečně odolný proti převodu null hodnoty a i proti neúspěšnému převodu. Vždy, když se nepovede text převést, vrací se výchozí zadaná hodnota

(defaultValue). Na každém místě zdrojového kódu, kde potřebujeme tento převod uskutečnit, se zavolá tato metoda a ušetří se tak vždy několik řádků kódu a zvýší se tak i celková čitelnost zdrojových kódů.

3.4 Spring Security framework

V době, kdy jsou informace jednou z nejcennějších hodnot, je velmi důležité řešit jejich zabezpečení. Lidé toužící po citlivých nebo neveřejných informacích, totiž neustále hledají způsoby jak data ukrást nebo získat. Proto je velmi důležité je patřičně chránit a zabezpečit proti neoprávněnému přístupu. K zabezpečení enterprise aplikace založené na Spring můžeme použít Spring Security framework.

3.4.1 Důvody použití Spring Security

Pro zabezpečení webové aplikace lze použít mnoho různých technik. Existují standardy jako ***Java Authentication and Authorization Service (JAAS)*** *nebo* ***Java EE Security.*** *Tyto standardy také (stejně jako Spring Security) zabezpečují autentizační nebo autorizační funkce. V tomto ohledu, je ale vítězem Spring Security, protože obsahuje tzv. top-to-bottom bezpečnostní řešení stručnou a rozumnou cestou. Proto je tento framework v oblasti bezpečnosti nejpoužívanějším mainstreamovým nástrojem pro řešení bezpečnosti.* [WINCH]

Spring Security framework obsahuje všechny výhody dependency injection a aspektově orientovaných technik. Pro použití Spring Security však není nutné aspekty psát ručně, framework tuto práci udělá za nás. Součástí Spring Security jsou obsáhlé řešení bezpečnosti, autorizace a autentizace, a to vše na úrovni vhodné pro webové aplikace.

3.4.2 Součásti frameworku

Spring Security framework je složen z několika modulů, kde každý modul má svoji funkci. Před prvním použitím frameworku je vhodné se s těmito moduly seznámit. Soupis jednotlivých modulů spolu s krátkým popisem je uveden v tabulce 3.

Modul	Popis
ACL	Poskytuje podporu pro objektovou bezpečnost skrz access control listy (ACLs).
Aspects	Menší modul pro aspekty založené na AspektJ.
CAS Client	Podpora pro samostatnou sign-on autentizaci, používající Jasig's Central Authentication Service (CAS).
Configuration	Obsahuje součásti pro konfiguraci Spring Security. Konfiguraci lze provádět v XML nebo Java třídě (od verze Spring Security 3.2).
Core	Modul poskytuje základní knihovny Spring Security.
Cryptography	Součástí modulu je kódování hesel (encryption) a šifrování.
LDAP	Modul pro podporu autentizace pomocí LDAP serverů.
OpenID	Obsahuje podporu pro centralizovanou autentizaci s OpenID.
Remoting	Podpora pro integraci se Spring Remoting.
Tag library	Spring Security knihovna pro JSP tagy.
Web	Bezpečnostní podpora pro webovou bezpečnost (filter-based).

Tabulka 3: Moduly frameworku Spring Security. [WALLS]

3.4.3 Bezpečnostní konfigurace frameworku

Spring Security framework obsahuje několik servlet filtrů používající různé aspektové metody zabezpečení. To ale neznamená, že je nutné konfigurovat každý takový bezpečnostní filtr zvlášť – stačí v konfiguračním souboru web.xml nastavit hodnotu DelegatingFilterProxy, který se o konfiguraci zbylých filtrů postará sám a není nutné tedy ostatní nastavení řešit jednotlivě.

```
<filter>
	<filter-name>springSecurityFilterChain</filter-name>
	<filter-class>
		org.springframework.web.filter.DelegatingFilterProxy
	</filter-class>
</filter>
```

Toto nastavení zjednodušeně znamená, že všechny příchozí požadavky budou zachyceny tímto filtrem a předány dále ke zpracování podřízeným filtrům.

Pro konkrétní konfiguraci bezpečnosti lze použít XML soubor nebo anotovanou Java třídu. Název XML souboru musí být zahnut v web.xml v části contextConfigLocation, konfigurační třída zase musí být potomkem (extends) třídy WebSecurityConfigurerAdapter.

V následujících částech je popsána konfigurace pomocí XML konfiguračního souboru.

Konfigurační soubor musí obsahovat hlavičku s definicí XML schémat a výchozího schématu.

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
xmlns:beans="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-4.0.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-
4.0.xsd">
```

Jako první schéma je uvedeno schéma security, které je zde uvedeno jako výchozí namespace – tagy z tohoto schématu se pak neuvádí s prefixem <security > ale zapisují se přímo bez prefixu. Jako minimální konfiguraci lze použít následující kód, který zabezpečí všechny stránky tak, aby na ně mohl přistupovat pouze uživatel s rolí ROLE_USER.

```
<http auto-config='true'>
      <intercept-url pattern="/**" access="ROLE_USER" />
</http>
```

V části intercept-url je definovaný pattern pro definování stránky a access pro omezení přístupu. V tomto případě tedy /** odpovídá jakékoli stránce, je tedy nutné pro přístup k dané stránce mít roli ROLE_USER. Častější varianta je, že intercept-url tagů definujeme více za sebou, stránka je pak zabezpečena podle toho tagu, který vyhovuje názvu (dle pattern) jako první. Z toho důvodu je také nutné, aby specifické nastavení bylo definováno na začátku a obecné nastavení až na konci definice.

Pro vložení uživatelských účtů se používá tag <authentication-manager>.

```
<authentication-manager>
      <authentication-provider>
            <user-service>
                  <user name="tomas" password="password"
                        authorities="ROLE_USER, ROLE_ADMIN" />
                  <user name="uzivatel" password="uzivpass"
                        authorities="ROLE_USER" />
            </user-service>
      </authentication-provider>
</authentication-manager>
```

Ukázkový kód zobrazuje základní nastavení uživatelských účtů. Častěji se však používají uživatelské účty načítané z databáze nebo LDAP serveru. Další ukázka obsahuje načtení

uživatelských účtů a rolí z databáze.

```
<authentication-manager>
      <authentication-provider>
            <password-encoder hash="bcrypt" />
            <jdbc-user-service data-source-ref="dataSource"
                  users-by-username-query=
            "select USERNAME, PASSWORD,  ENABLED" +
                  "from USERACCOUNTS " +
                  "where USERNAME = ? and KOOPERACNI_PORTAL = '*'"
                   authorities-by-username-query=
            "select USERNAME, ROLE from USERROLES " +
            "where USERNAME = ?"/>
      </authentication-provider>
</authentication-manager>
```

V ukázce je také použit tag <password-encoder> s parametrem hash, který obsahuje hodnotu použitého hashování hesla uloženého v databázi. Místo předchozího <user-service> je zde použit <jdbc-user-service> s referencí na dataSource pro připojení k databázi a zadanými dvěma dotazy (na ověření uživatelského účtu a hesla, a na získání uživatelských rolí).

Již zmíněný tag <http auto-config='true'> může obsahovat následující elementy a jejich parametry:

- pro definování přístupu k jednotlivým stránkám a použitému zabezpečení (https)

```
<intercept-url pattern='/main/*' access='ROLE_USER'
      requires-channel="https">
```

- pro definování logovací stránky a hlavní stránky po přihlášení

```
<form-login login-page='/login.hml'
      default-target-url='first_page.html'
      always-use-default-target='true' />
```

- stránka po ohlášení, odkaz pro odhlášení a kroky po odhlášení (smazat cookies JSESSIONID a zneplatnit celou session)

```
<logout-success-url="/index.html?logout"
      logout-url="/logout"
      delete-cookies="JSESSIONID"
      invalidate-session="true" />
```

- pro nastavení výchozích portů jednotlivých protokolů

```
<port-mappings><port-mapping http='8080' https='8443'
```

- pro přesměrování na stránku při neplatném session ID

```
<session management invalid-session-url="/sessionTimeout.html" />
```

- tag pro omezení počtu přihlášení uživatele, pro správnou funkci je potřeba do web.xml přidat listener org.springframework.security.web.session.HttpSessionEventPublisher

```
<session-management>
<concurrency-control max-sessions="1" error-if-maximum-exceeded="true">
```

- pro logování pomocí OpenID technologie, elementy v tomto atributu nastavují konkrétní atributy poskytovatele OpenID

```
<openid-login>
```

- pro povolení funkce CSRF protection, která slouží proti útokům typu Cross site request forgery

```
<csrf />
```

Pro vlastní obsluhu chybových hlášení Spring Security frameworku lze použít uvnitř elementu <form-login> atribut authentication-failure-handler-ref="..." s referencí na beanu obsahující vlastní mapování výjimek. Příklad takové beany je v následující ukázce zdrojového kódu:

```
<beans:bean id="customFailureHandler"
class="org.springframework.security.web.authentication.ExceptionMapping
AuthenticationFailureHandler">

      <beans:property name="exceptionMappings">

            <beans:props>

                  <beans:prop
key="org.springframework.security.authentication.BadCredentialsExceptio
n">/login.html?error=1
                  </beans:prop>

            </beans:props>

      </beans:property>

      <beans:property name="defaultFailureUrl"
            value="/login.html?defaulterror" />

</beans:bean>
```

V ukázce je namapována chyba přihlášení z důvodu špatně zadaných údajů, která je přesměrována na přihlašovací stránku login.html s parametrem error nasataveným na hodnotu 1. Jako výchozí url pro nespecifikované chyby je uvedena stejná stránka login.html, ale s parametrem defaulterror. Obsluhu těchto jednotlivých parametrů a patřičné zobrazení textu výjimky si už řeší zobrazovací komponenta login.html stránky sama.

Ochrana webové aplikace proti útokům

Následující podkapitoly popisují vybrané dva typy útoků na webové aplikace. Útoky těchto dvou typů umí Spring Security úspěšně odhalit a aplikaci tak ochránit. Je ale nutné, aby ochrana nebyla ručně vypnuta, proto je také dobré, aby vývojář aplikace o těchto typech útoků věděl a nedopatřením nebo nevědomostí ochranu nevypnul. Je také dobré si zjistit, zda daná verze Spring Security frameworku má ochranu ve výchozím nastavení zapnutou, nebo je ji nutné nastavit v konfiguraci.

Cross site request forgery

Mezi potencionální nebezpečí patří útok typu cross site request forgery. CSRF je typ útoku, kdy oběť odesílá nevědomě požadavky na určitý server bez jejího vědomí. Odesílání požadavků se odehrává v rámci identity a oprávnění napadaného uživatele. Většina webových stránek totiž spolu s odesílaným požadavkem odesílá (pro identifikaci přihlášeného uživatele) také informace o session cookies, IP adrese nebo jiných potencionálně užitečných informací. Proto, pokud je uživatel v současné době přihlášen na určité webové stránce, server nemá možnost jak rozlišit požadavky zaslané vědomě uživatelem od požadavků, které sice přišli od uživatele, ale zaslat je vědomě nechtěl. Tento typ útoku lze použít na odesílání nevyžádaných příspěvků, ale třeba také na odeslání požadavku na změnu přihlašovacího mailu nebo hesla. [OWASP]

Příkladem může být stránka, kde má uživatel možnost přidávat svoje příspěvky (diskuse, fóra, recenze, apod.). Pro přidání příspěvku je nutné volat stránku /addComment metodou POST s potřebnými parametry komentáře (jméno, datum, text zprávy, apod.). Stránka se volá přes url www.example.cz/addComment. Útočník pak může mít na svojí vlastní stránce nebo webu html kód podobný tomu v následující části.

```
<form method="POST" action="http://www.example.cz/addComment">

	<input type="hidden" name="message"
		value="Zprava z ciziho neovereneho zdroje!" />

	<input type="submit" value="Odeslat" />

</form>
```

V případě že je uživatel na stránce example.cz již přihlášený a klikne na takto připravený odkaz nebo formulář, může nevědomě přidat příspěvek pod svým jménem, aniž by o to stál. Pokud by se jednalo například o nějakou sociální síť, kde je pravděpodobnost přihlášení větší než u stránky example.cz, tak lze tímto způsobem například sdílet reklamy nebo odkazy pod přihlašovacím jménem uživatele, který o to nestál.

Spring Security implementuje ochranu před typem útoku CSRF pomocí synchronizačního tokenu. Všechny „state-changed“ požadavky (všechny požadavky na server kromě GET, HEAD, OPTION nebo TRACE) jsou zachyceny a označeny na kontrolu CSRF tokenu. Pokud

tyto požadavky token neobsahují nebo token nesouhlasí s tokenem na serveru, aplikace vrátí výjimku o chybném CSRF tokenu (CsrfException). Pro správné odesílání veškerých požadavků na serveru je tedy nutné, aby požadavek obsahovat _csrf pole s hodnotou stejnou jako je vygenerovaná hodnota na serveru. Přidání bezpečnostního pole do formuláře můžeme provést například pomocí tagu input.

```
<input type="hidden" name="${_csrf.parameterName}"
      value="${_csrf.token}" />
```

Výše zmíněný příklad útočníkovi stránky s falešným formulářem nemůže obsahovat správné hodnoty CSRF tokenu, protože jej nezná, a proto server takový požadavek na zpracování formuláře správně odmítne. CSRF ochrana je od verze Spring Security 3.2. defaultně zapnutá a není tedy nutné ji manuálně zapínat v konfiguračním souboru. (volně převzato z [WALLS])

Session fixation

U větších webových projektů je potřeba si uložit přihlášení uživatele, aby se nemusel na každé stránce prokazovat platnými přihlašovacími údaji. K tomu slouží mechanismus uložení Session ID (SID) do tzv. cookies, pomocí kterých se pak ověřuje, zda je uživatel přihlášen a má přístup k požadovanému obsahu (server ověřuje zda Session ID u uživatele odpovídá Session ID při přihlášení uživatele).

Útok na tento mechanismus spočívá v tom, že útočník pošle oběti odkaz, který obsahuje přednastavené SID s jemu známou hodnotou. Pokud oběť na odkaz klikne, přihlásí se pod svými přihlašovacími údaji a odešle přihlašovací údaje na server spolu s podvrženým SID, server pak nemá možnost ověřit, zda si zabezpečený obsah prohlíží oběť nebo útočník, protože oba jsou ověřeny pomocí stejného SID. Existuje několik variant tohoto útoku podvržením SID, ať už popsaná verze podstrčení SID (útočníkovi známá hodnota), nebo varianta, kdy útočník zjistí uživatelské SID a sobě si nastaví stejné. [MCNAB]

Spring Security framework se proti tomuto útoku brání tak, že SID uživatele zneplatňuje a při novém přihlášení vždy vytvoří nové SID. Tato varianta je nastavena jako výchozí. Pokud by z nějakého důvodu vytváření nového SID bylo potřeba vypnout, lze použít v konfiguraci element <session-management> s atributem session-fixation-protection a následujícími hodnotami:

- migrateSession – výchozí hodnota, vytváří novou session a kopíruje existující atributy z původní do této nově vytvořené session,
- none – neudělá nic, existující session bude zachována a aplikace je potencionálně zranitelná právě útokem na podvržení session (pokud není proti útoku zabezpečena jinak),
- newSession – vytvoří novou session, ale nekopíruje atributy z původní do této nové.

3.4.4 Omezení přístupu k metodám

Spring Security framework umožňuje vedle zmíněného zabezpečení (konfigurovatelného v konfiguračním souboru nebo třídě) také zabezpečit jednotlivé metody přímo ve zdrojovém kódu dané metody. Tato možnost zabezpečení se musí nastavit v konfiguračním souboru elementem, který tuto možnost povolí:

```
<global-method-security pre-post-annotations="enabled" />
```

Omezení přístupu k metodě (její volání) se pak provádí pomocí anotace @PreAutorize s definovanou hodnotou omezení přístupu.

```
public interface RozhraniTridy2 {

      @PreAutorize(„hasAuthority('ROLEADMIN')")
      public String getName();

      @PreAutorize(„isAuthenticated()")
      public String findByName(String name);

}
```

V příkladu je metoda getName povolena spouštět pouze pro přihlášené uživatele s rolí ROLEADMIN, v případě metody findByName se jedná o jakkoliv přihlášeného uživatele.

3.5 Hibernate

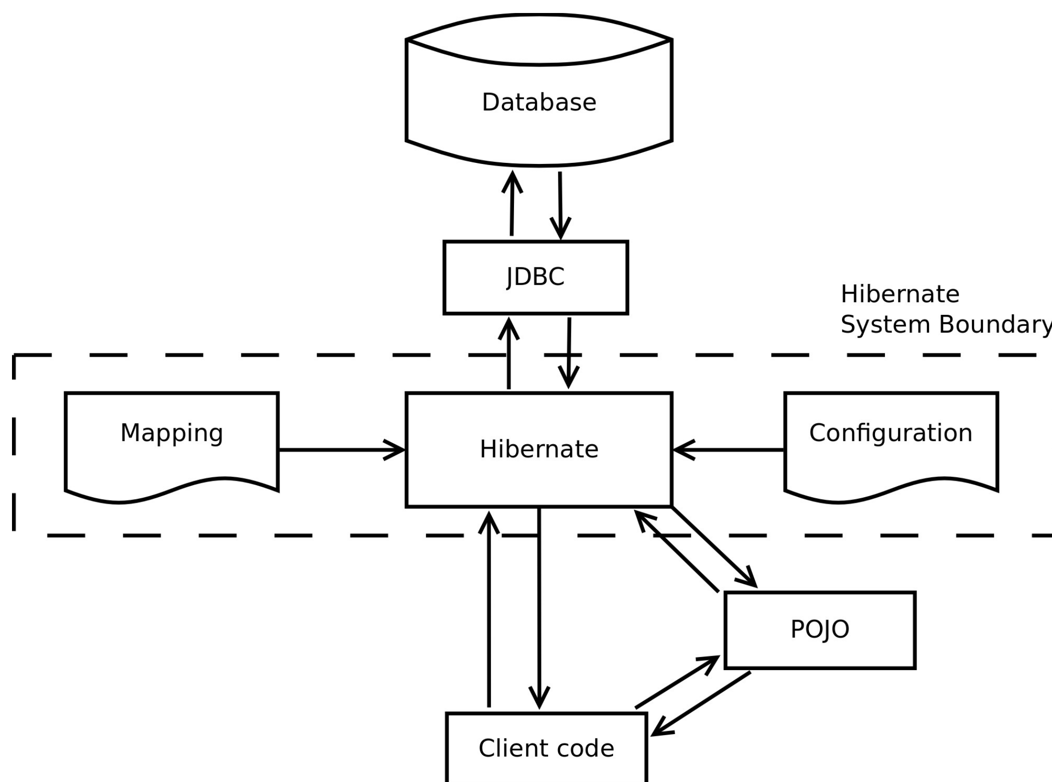
Další z popisovaných technologií slouží pro ukládání objektových dat z Java kódu do relační databáze. Díky objektovému programování jsou tu pak nástroje typu Hibernate pro ukládání těchto objektů do relačních databází. Obecně tyto nástroje patří do kategorie ORM (Objektově relační mapování).

Knihovna Hibernate umožňuje snadné použití relačních databází v Java aplikacích, protože nám umožňuje snadno nastavit můstek pro převod relačních dat na jednoduché Java objekty a obráceně. Při vývoji aplikací je snaha o co nejvěrnější reprezentaci reálných věcí, v jazyce Java často nazývaných jako entity. Tyto entity jsou pak přes ORM převedeny na řádky relačních tabulek pomocí nastaveného mapování. Hibernate se snaží do jisté míry odstínit vývojáře od psaní SQL dotazů. Místo klasického SQL je v knihovně Hibernate dostupný dotazovací jazyk HQL, který je jakousi náhradou klasických SQL dotazů. Spojení knihovny Hibernate a relační databáze se děje (stejně jako v klasické Java aplikaci) pomocí JDBC vrstvy, která je zprostředkovatelem spojení a přístupu k relační databázi. [OTTINGER]

Představa ideálního vývoje a významného zjednodušení konfigurace ukládání dat by mohla vypadat podobně jako následují ukázka. Stačilo by vytvořenou instanci jakékoli třídy nebo entity vložit jako parametr metody pro ukládání dat, metoda by se pak sama postarala o uložení do databáze a to bez nutnosti další konfigurace.

```
Pracovnik p = new Pracovnik(...);
UkladaciTrida u = UkladaciTrida.getInstance();
u.save(p);
```

Takové zjednodušení by bylo velice příjemné, ale v reálném vývoji by narazilo na mnoho nutných konfigurací a nastavení. Knihovna Hibernate toto téměř jednoduché ukládání objektů umožňuje, nicméně je nutné provést některé nutné kroky před tím, než je taková část zdrojového kódu funkční. Na obrázku 9 je graficky znázorněna role knihovny Hibernate jako zprostředkovatele databázového obsahu. Z obrázku je patrné, že knihovna Hibernate komunikuje s databází pomocí JDBC rozhraní (tzn. přes klasické SQL dotazy), které Hibernate knihovna sama generuje. Klientský kód zase komunikuje s knihovnou Hibernate, buď pomocí zmiňovaného HQL jazyka nebo pomocí metod určených právě k ukládání, aktualizaci nebo i získávání objektů.



Obrázek 9: Znázornění role knihovny Hibernate v Java aplikaci.[OTTINGER]

Ke správném uložení objektu je nutné mít správně nastavené mapování dané entity. Pomocí

nastaveného mapování vlastně určujeme, který objekt patří ke které tabulce, dále pak který atribut daného objektu patří ke kterému sloupečku relační tabulky. Mapování lze provádět jak pomocí XML, tak pomocí anotací. Pomocí anotací lze mapování třídy popsat přímo v dané třídě, z tohoto důvodu jsou zde ukázky anotace a ne XML verze mapování. Existuje také situace, kdy explicitně nemusíme u každé třídy a každého jejího atributu definovat, ke které tabulce nebo sloupečku tabulky patří – jedná se o situaci, kdy máme třídy pojmenované stejně jako tabulky, do kterých daná třída patří. V extrémním případě lze dokonce říci, že vývojář vůbec nemusí vědět, do jaké tabulky se objekt uloží nebo z jaké tabulky se objekt čte a tuto volbu nechá na technologií Hibernate. Tato situace se ovšem netýká zde popisovaného webového portálu, protože zde jsou objekty mapovány na již existující relační strukturu dat. Následující část ukázkového kódu zobrazuje příklad nastavení mapování třídy Osoba na tabulku se jménem PERSON.

```
@Entity
@Table(name = "PERSON")
public class Osoba {

      @Id
      @Column(name = "PERSON_ID"
      int idOsoba;


      @Column(name = "PERSON_NAME")
      String name;

}
```

Pro uložení instance této třídy do databáze pomocí Hibernate lze pak použít metodu persist třídy Session. Kompletní část procesu ukládání včetně vytvoření instance třídy Osoba je naznačena v následující ukázce.

```
...

@Autowired
SessionFactory factory;

…

@Test
public void saveOsoba(String name) {

      Osoba o = new Osoba(name);

      Session session = factory.openSession();

      Transaction trans = session.beginTransaction();

      session.persist(o);

      trans.commit();

      session.close();

}
```

Podobně lze ukázat metodu pro získání seznamu osob. Zde na rozdíl od předchozí ukázky je použit dotazovací jazyk HQL.

```
public List<Osoba> getOsoby() {
      Session session = factory.openSession();
      List<Osoba> osList =
            (List<Osoba>) session.createQuery("from Osoba").list();
      session.close();
      return osList;
}
```

3.5.1 Hibernate Query Language

Jazyk HQL je dotazovací jazyk podobný SQL, na rozdíl však od SQL je plně objektově orientovaný s podporou vztahů mezi třídami (dědičnost, asociace, polymorfismus). HQL stejně jako SQL není tzv. case-sensitive, nicméně u volání Java tříd je nutné zachovat velké písmenko v názvu třídy. V předchozím příkladu na získání seznamu osob z databáze je použit dotaz „from Osoba“ kde je právě použit název třídy Osoba. Stejně by tedy fungoval i příkaz „FROM Osoba“, nicméně dotaz „from osoba“ by již měl problém dohledat správnou třídu, proto by nevrátil požadovaná data a HQL jazyk by vrátil chybové hlášení. V HQL dotazech se lze také setkat s názvem třídy v kompletní podobě, tzn. včetně kompletní cesty ke třídě ve struktuře balíčků (např. cz.firma.model.Osoba). Tato varianta se používá, pokud je zakázaná funkce auto-import a je tedy nutné třídy pojmenovávat včetně cesty k nim.

V pokročilejších dotazech je také nutné třídy spojovat pomocí „join“ podobně jako v SQL. Zde je nutné zdůraznit, že pro přístup k atributům třídy v HQL je nutné používat aliasy. Následující ukázka zobrazuje složitěji konstruovaný HQL dotaz na získání osoby a její adresy.

```
from Osoba o
left outer join o.adresa as adresa
where o.idOsoba = '%s'
```

Standardně podporovaná spojení jsou levé/pravé vnější (left outer join, right outer join), vnitřní spojení (inner join) a plné spojení (tzv. full join).

Pokud je potřeba získat konkrétní údaj (atribut), je možné dotaz rozšířit o část „select“, kde se přesně definují atributy pro dotaz. Příkladem může být získání věku konkrétní osoby.

```
Select o.age from Osoba o where o.idOsoba = 52941
```

HQL také podporuje agregační funkce známé z SQL jazyka, lze tedy použít avg(...), min(...), max(...), sum(...), count([distinct|all] …) s agregačními funkcemi souvisí podpora group by a having se stejným způsobem použití jako v SQL. Pro sčítání numerických hodnot lze použít

známé aritmetické operátory + - * /, pro zjištění prázdných hodnot atributů lze použít is null nebo is not null a mnoho dalších variant syntaxe a funkcí známé z jazyka SQL (<=, =>, <>, ! =, like, between, is empty, in, not in, ||, nullif() …). Pro případ nutnosti třídění výsledných dat je možné použít order by s definicí pořadí sloupečků a s podporou asc nebo desc třídění.

Pro ruční psaní update, insert nebo delete příkazů má HQL stejnou syntaxi jako SQL. Pro použití těchto příkazů se nejčastěji používá metoda executeUpdate() třídy Query. Následující ukázka zdrojového kódu zobrazuje jednoduchou změnu jména ve třídě Osoba pomocí Hibernate knihovny.

```
Session session = sessionFactory.openSession();
Transaction trans = session.beginTransaction();
String hql = “update Osoba o “ +
                 “ set o.name = :jmeno “ +
                 “ where o.idOsoba = :id”;
Query q = session.createQuerry(hql);
q.setString(“jmeno”, “Tomas Beran”);
q.setInt(“id”, 97541);
q.executeUpdate();
tx.commit();
session.close();
```

3.6 Firebird SQL databáze

Datovým úložištěm popisované webové aplikace je Firebird SQL databáze. Důvod volby této databáze byl již zmíněn v úvodu této práce (stávající informační systém používá tuto databázi k ukládání veškerých dat a webový portál potřebuje tyto data číst).

Firebird je relační databáze podporovaná napříč různými platformami operačních systémů (Windows, Linux, Mac Os a různé varianty Unix). S touto databází se lze setkat v různých produkčních systémech již od roku 1981 (od té doby pod různým označením a jmény). Firebird Project je komerčně nezávislý projekt C a C++ vývojářů, techniků a ostatních lidí, pohybujících se kolem vývoje databáze. V oficiálních zdrojích (firebird.org) a dokumentacích si vývojáři zakládají na tom, že open-source projekt Firebird je zdarma i pro komerční použití nebo distribuci s vlastním komerčním systémem. [FIREBIRD]

Poslední oficiální uvolněná serverová verze nese označení Firebird 2.5.5 (informace z ledna 2016). Nicméně již déle než rok je k dispozici verze Firebird 3, která ale nese stále označení jako beta. Tato verze obsahuje velké množství změn a nových funkcionalit oproti stabilní verzi 2.5, nicméně pro produkční prostředí se stále více hodí stabilní verze.

Mezi klíčové funkce a vlastnosti Firebird databáze patří:

- Podpora psaní vlastních procedur a triggerů, která umožňuje rozšiřovat integritní omezení a psát vlastní logiku aplikace na úrovni databáze.
- Plná podpora koncepce ACID v transakcích (Atomicity, Consistency, Isolation, Durability). To znamená, že každá transakce je buď celá úspěšně dokončena nebo celá odvolána a nevznikají tak stavy, kdy je uložena pouze část požadovaných změn. Data jsou také vždy v plně konzistentním stavu, pro jednotlivé transakce izolována s garancí, že jednou úspěšně provedená změna bude do databáze zapsána. Není tedy možné aby k zápisu nedošlo, například z důvodu havárie nebo výpadku databázového stroje.
- Podpora multigenerační architektury (Multi Generation Architecture).
- Podpora vlastních funkcí (UDF – User defined function), která umožňuje dopsat si do databázového stroje vlastní uživatelskou funkci, která se ve standardních funkcích nevyskytuje.
- Možnost automatického generování hodnot pomocí generátorů k hodnotám v tabulkách nebo generování unikátního ID klíče do tabulek, zajišťování referenčních integrit, plná podpora indexů.
- Plná řada podpůrných nástrojů:
    - pro prohlížení dat např. Flamerobin, Turbobird nebo IBAccess
    - pro získávání výkonnostních a analytických informaci – IBAnalyst, FBScanner,
- Propracovaná metodika přístupových práv, díky které lze řešení přístupových práv (read, insert, update, delete, execute…) v aplikaci řešit přímo na úrovni databáze.

3.6.1 Multi Generation Architecture

V předchozí části byla zmíněna podpora MGA ve Firebird databázích. Jedná se vlastně o systém, který nahrazuje zamykání hodnot v tabulkách při transakčním zpracování.

Dřívější metody transakčního zpracování nastavovaly na čtené nebo měněné hodnoty zámky, které ostatním transakcím říkaly, že data mění někdo jiný a nelze je tedy měnit. Zamykání mohlo probíhat v extrémním případě na celou tabulku (při updatu jednoho řádku zcela zbytečné a omezující), nebo se uzamkl aktuálně měněný řádek. Nejlepší variantou zamykání pak bylo zamykání pouze měněného údaje (tzn. konkrétní řádek a konkrétní hodnota – sloupec). Stavy jednotlivých zámků se pak ukládaly do speciálních souborů mimo databázový stroj pro případ pádu databázového stroje.

Architektura MGA metodu zámků nahrazuje a pro každý jednotlivý záznam uchovává hned několik verzí záznamu s číslem transakce a stavem transakce. Tento záznam se z angličtiny nazývá Data page. Každá transakce je identifikována pomocí unikátního čísla, kterou jí přiděluje databázový stroj. Toto číslo pak právě figuruje v jednotlivých Data page záznamech. Vedle Data page záznamů také databázový stroj obsahuje záznamy o stavech jednotlivých transakcí – TIP (Transaction Inventory Pages), jedná se o seznam čísel transakcí (0 ... poslední číslo transakce) a jejich stavů. [IB-EXPERT]

Transakce ve Firebird databázi může nabývat následujících stavů:

- 00 – Active,
- 01 – Commited,
- 10 – Rolled back,
- 11 – Limbo (stav transakce při chybě při dvou fázovém zpracování transakcí).

Následující tabulka 4 zobrazuje informace v TIP, kde je naznačeno několik prvních transakcí a jejich aktuálních stavů zpracování.

Transaction Invetory Pages	
Transaction number	**Transaction state**
0	Committed
1	Committed
2	Committed
3	Committed
4	Rolled back
5	Committed
6	Committed
7	Rolled back
8	Active
9	Committed
10	Active

Tabulka 4: Ukázka záznamů o stavu jednotlivých transakcí TIP. [KUZMENKO]

Data page		
No.	**Transaction number**	**Data**
1	2	100
2	6	150
3	7	200

Tabulka 5: Ukázka záznamů verzí jednoho konkrétního záznamu v tabulce. [KUZMENKO]

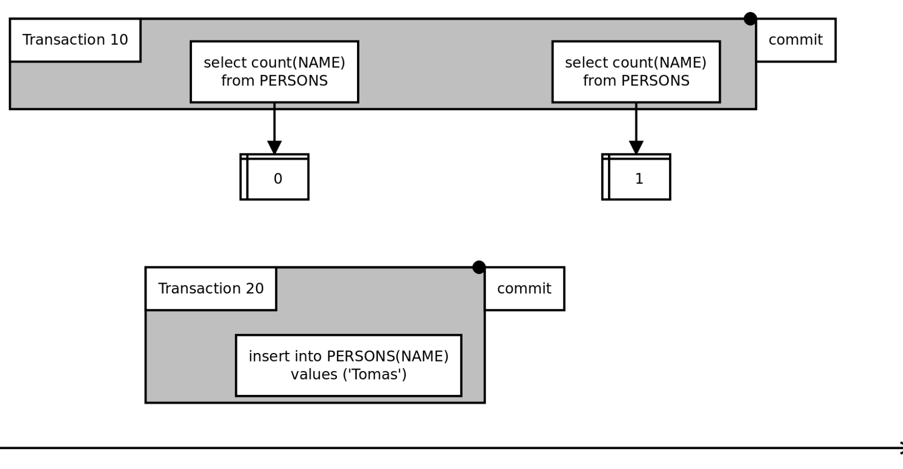
Pokud uvažujeme stejné číslování transakcí v tabulce 4 a 5, tak lze prohlásit, že transakce číslo 2 uložila hodnotu 100 do databáze, transakce číslo 6 pak opět změnila hodnotu na 150 a transakce číslo 7 se o změnu pokusila, nicméně byla ukončena stavem Rolled back – změna nebyla potvrzena. Pokud bychom uvažovali, že transakce číslo 8 požaduje hodnotu záznamu, dostane zpět data 150, protože poslední předchozí potvrzená verze (commit) dat pochází od transakce číslo 7 a je s hodnotou 150.

Při transakčním zpracování vznikají klasické problémové situace, kdy se dvě transakce snaží měnit jeden a ten samý záznam. V dříve popisované variantě zamykání záznamů tomu bylo zabráněno tak, že transakce před změnou nastavila editovací zámek na určitý záznam a ostatní transakce tak přesně věděly, že daný záznam nelze editovat (vznikl by update conflict). V MGA architektuře je tomu zabráněno v okamžiku pokusu o potvrzení transakce (commit), kdy v případě transakce s nižším číslem a potvrzenou změnou nelze záznam editovat (deadlock - update confict with concurrent transaction).

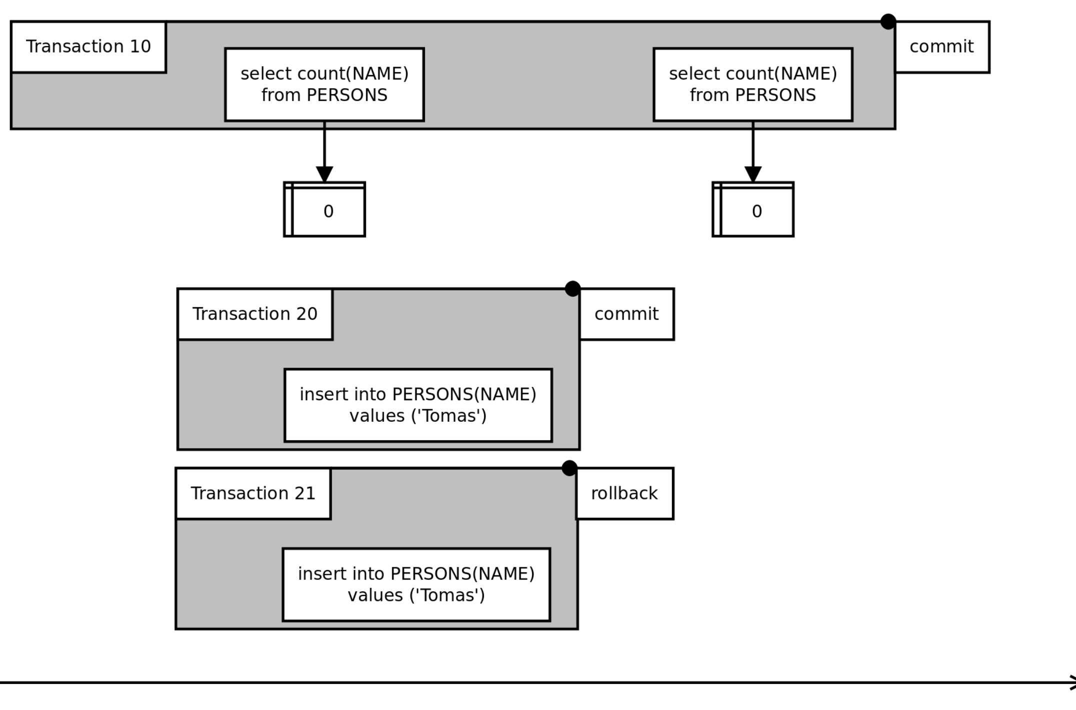
Existuje několik variant, jak transakce mezi sebou čtou editovaná data. Ve Firebirdu jsou standardně použity varianty nazývané Read Commited nebo Snapshot. Rozdíl mezi těmito variantami je v tom, že varianta Read Commited čte informace o stavu transakcí z globálního TIP záznamu, ale varianta Snapshot si na startu transakce zkopíruje TIP záznam a stav transakcí čte z této kopie. Otázkou tedy zůstává, jaká data jednotlivé typy transakcí vidí - samozřejmě vidí svoje vlastní změny. Pokud se jedná o Read Commited, tak transakce vidí i všechny aktuálně potvrzené změny v okamžiku dotazu, je to proto, že stav transakcí čte z globálního TIP záznamu. Pokud se jedná o variantu Snapshot, tak transakce vidí všechny potvrzené změny, které byly potvrzeny před samotným startem transakce. Transakce si totiž zkopírovala TIP záznam při zahájení a změnu stavu transakcí v průběhu nevidí.

Obrázek 10 zobrazuje dvě transakce číslo 10 a 20, kdy transakce číslo 10 začala dříve a v první fázi čte počet záznamů z tabulky PERSON – ve chvíli čtení záznamu nejsou v tabulce žádná data a proto je výsledkem hodnota 0. Během života transakce číslo 10 vznikla nová transakce s číslem 20, která záznam do tabulky PERSON vkládá a okamžitě potvrzuje vložená data (commit). Pokud poté transakce číslo 10 provede znovu dotaz na počet záznamů, databáze již vrátí hodnotu 1, protože porovnáním globálního záznamu TIP a Data Page v tabulce zjistí, že nová verze od transakce 20 již byla potvrzena a je proto platná.

Obrázky 10 a 11 mají v dolní části naznačenou časovou osu, kde čas plyne zleva doprava.



Obrázek 10: Transakce typu Read Commited a čtení potvrzených dat. [autor]



Obrázek 11: Transakce typu Snapshot a čtení potvrzených dat. [autor]

Na rozdíl od předchozího obrázku 10 (s transakcí typu Read Commited) obrázek 11 zobrazuje transakce nastavené jako Snapshot. Transakce číslo 10 si před zahájením zkopíruje stav jednotlivých transakcí TIP (kde nefigurují transakce číslo 20 ani 21, maximální číslo transakce v tabulce je 9). Při čtení počtu záznamů z tabulky PERSONS se opět porovnávají záznamy Data page a TIP. Záznamy o stavech transakcí jsou ale z kopie globální tabulky v okamžiku startu transakce 10, proto databáze vyhodnotí, že žádná potvrzená transakce

záznam nevložila a je tedy během celé doby běhu transakce číslo 10 vracen stále stejný výsledek na dotaz o počtu záznamů. Výjimku by tvořila situace, kdy sama transakce 10 vloží záznam, pak by nemuselo platit tvrzení, že transakce dostává stále stejný výsledek na stejný dotaz.

Typ jednotlivých transakcí nastavujeme jak globálně, kdy máme nastavené spojení na určitý výchozí typ transakce, tak i jednotlivě pro každou transakci, kdy můžeme pomocí SET TRANSACTION nastavit požadovaný typ transakce. Existují také jiné varianty transakčního zpracování (např. Read uncommited – vzájemné čtení nepotvrzených změn), ty ale nejsou Firebird databází plně podporovány a proto zde nejsou podrobněji popsány.

Existuje také možnost v jedné transakci spustit nezávisle jinou transakci. Ta se nazývá autonomous transaction a je nezávislá na rodičovské transakci. Tzn. změny potvrzené v autonomní transakci zůstanou potvrzené i v situaci, kdy rodičovská transakce nebude potvrzena. Autonomní transakce lze volat z triggerů, procedur nebo samostatných bloků a je vhodná například pro logování pokusů o změnu nebo vyhledávání chyb ve složitější struktuře triggerů a procedur. Následuje ukázka takové transakce, kde sledujeme jakýkoliv pokus o smazání záznamu z tabulky PERSON i přesto, že to není povoleno. Na triggeru PERSON_BD je při jakémkoli pokusu o smazání záznamu volána výjimka o nemožnosti mazání záznamů. Jakákoli transakce s pokusem smazat záznam tabulky PERSON tedy skončí chybou a stavem rolled-back, nicméně záznam do tabulky LOG_TABLE se uloží, protože je vložen v jiné (autonomní) transakci.

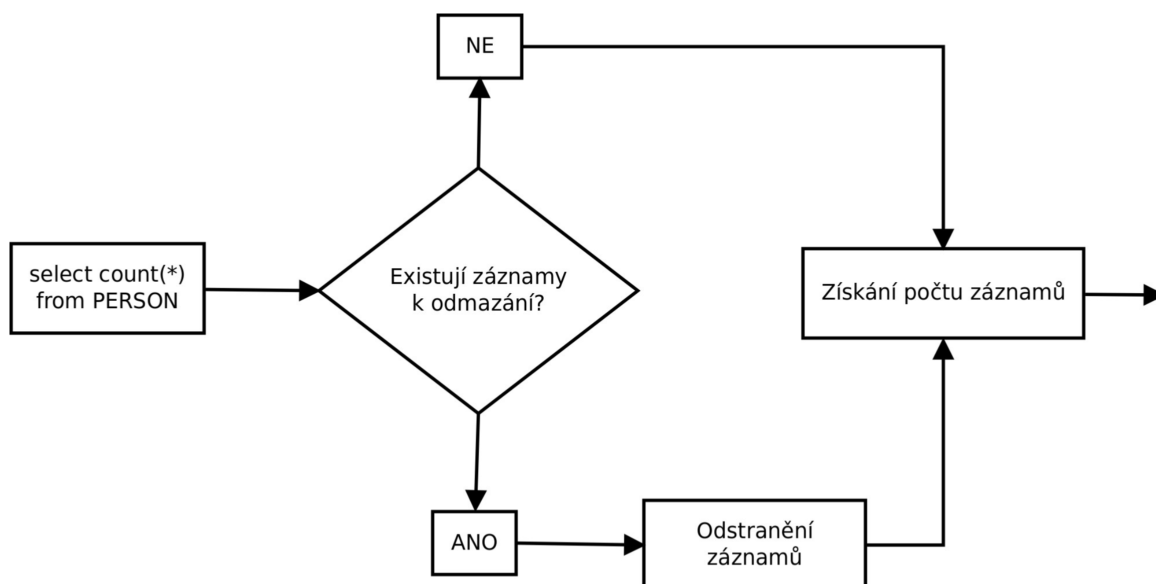
```
create or alter trigger PERSON_BD for PERSON before delete do
begin
    in autonomous transaction do
        insert into LOG_TABLE (LOG)
            values ('Uzivatel '|| current_user || ' se pokusil'
                || 'smazat záznam ' || OLD.NAME);
    exception ex 'Mazání záznamů není povoleno';
end
```

3.6.2 Garbage collection

Při psaní této kapitoly bylo čerpáno z informací poskytnutých na konferenci Firebird tour 2013 [KHORSUN]. U každého záznamu jsou tedy evidovány jednotlivé verze (transakce, které záznam měnily), nicméně pokud by se evidovaly skutečně všechny změny, tak by velikost databáze po čase začala neúnosně narůstat a tím by se snižoval celkový výkon databázového stroje (pracoval by na stále větší množinou záznamů). Proto ve Firebirdu (podobně jako v jiných technologiích) existuje nástroj na uvolňování paměti – tzn. nástroj pro odmazávání nepotřebných záznamů nazývaný obecně Garbage collection (dále také GC).

Jeho funkci lze popsat tak, že se stará o odmazávání nepotřebných Data page záznamů. Nepotřebné záznamy lze definovat jako záznamy, které žádná aktivní transakce nemůže vidět (protože existuje novější verze). Databázový stroj si pro jednoduchost a rychlost identifikace takových záznamů pamatuje číslo nejstarší aktivní transakce (Oldest Snapshot Transaction – dále také zkráceně OST). Toto číslo transakce určuje nejstarší verzi Data page záznamu, který ještě nemůže být odmazán (transakce se k němu může ještě vrátit). Pokud je tedy nějaký záznam starší než OST, je možné ho smazat z Data page záznamů. Stejně tak může Garbage Collection odstranit záznamy, které byli vytvořeny Rolled-back transakcemi.

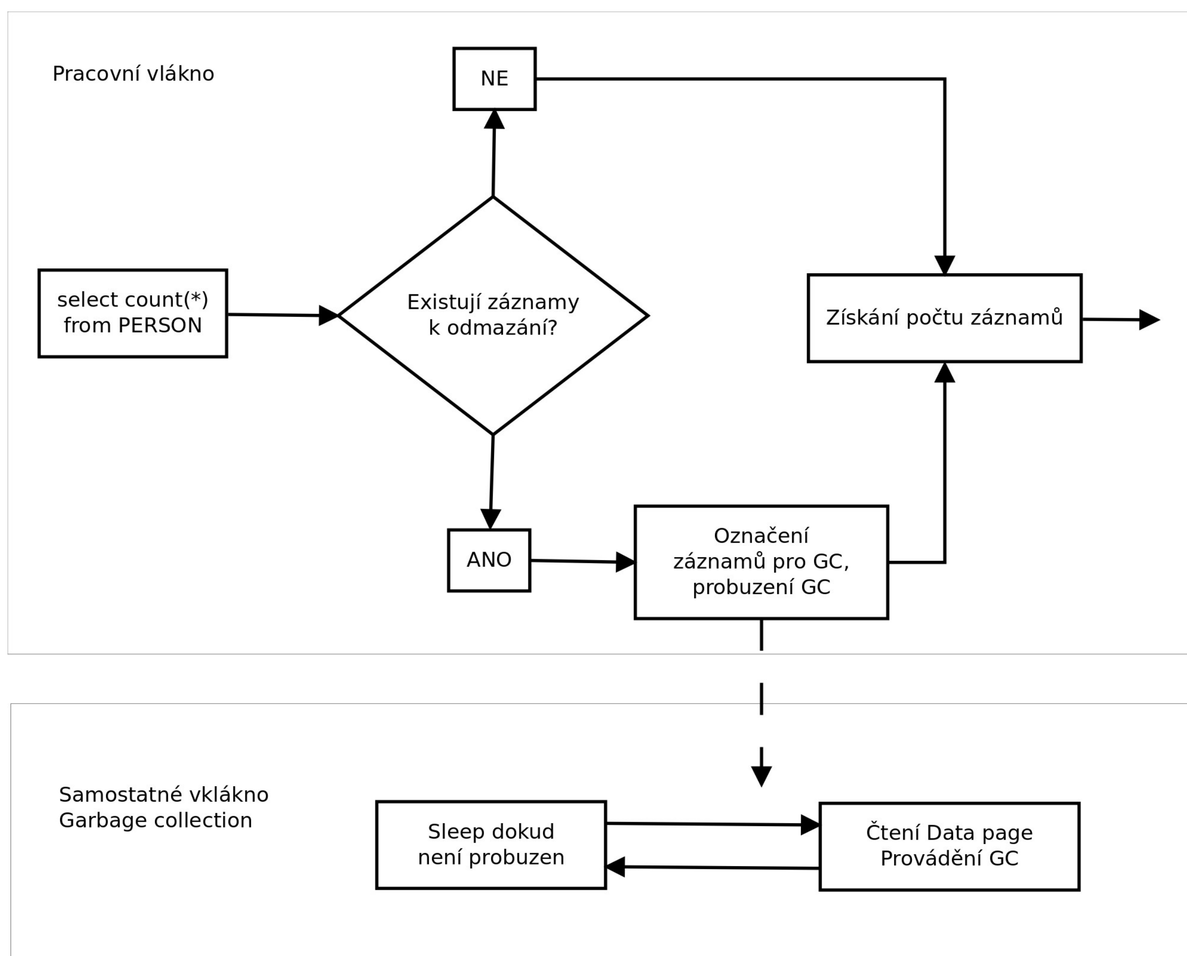
Před samotným odmazáním nepotřebných záznamů je nutné tyto záznamy najít. Garbage Collection pro svojí paměťovou náročnost v produkčních prostředích nevyhledává tyto záznamy aktivně sám. Záznamy k odmazání jsou kontrolovány před každým požadavkem na čtení dat, kdy tyto záznamy jsou vyhodnocovány a případně odmazány, pokud splňují podmínky pro smazání. Odmazání takto identifikovaných záznamů může proběhnout okamžitě, tato varianta se nazývá Cooperative Garbage Collection (data jsou z Data page skutečně okamžitě odmazána) nebo tyto záznamy jsou pouze označeny k odmazání a smazány budou později (varianta Background Garabage Collection). Na obrázku 12 je pomocí vývojového diagramu naznačen tento typ odmazávání nepotřebných verzí záznamů.



Obrázek 12: Cooperative Garbage Collection - metoda přímého odmazávání nepotřebných záznamů. [KHORSUN]

Druhou variantou je, že se data pouze označí k prozkoumání GC a jejich kontrola a odmazání pak probíhá v separovaném vlákně, tato varianta se také označuje jako Background Garbage Collection. Její fungování je naznačeno na obrázku 13 s vývojovým diagramem, kde jsou zobrazeny dvě samostatná vlákna (pracovní a vlákno GC). Vlákno s GC se probouzí až v

případě, že nějaké pracovní vlákno zjistí výskyt nepotřebných záznamů k odmazání.



Obrázek 13: Background Garbage Collection - odmazávání starých záznamů v separovaném vlákně. [KHORSUN]

Ve větších serverových databázích se zpravidla Cooperative GC vůbec nepoužívá, protože celkově může zpomalovat rychlost čtení záznamů. Naopak například v single user databázi na lokální stanici, kdy se databáze používá pouze pro ukládání jednoho nebo několika málo uživatelů, může samostatné vlákno GC zbytečně zabírat procesorový čas a je proto vhodnější zvolit Cooperative GC.

Problém s těmito GC může nastat, pokud vznikne mnoho Data page záznamů, které delší dobu nikdo nečte (například se tam provádí pouze update transakce). Tyto záznamy se nikdy nedostanou na seznam záznamů pro GC a v databázi mohou zůstávat zbytečně dlouho. Řešení jak této skutečnosti zamezit je hned několik:

- Číst obsah celé databáze (postupně provést select * from … ze všech tabulek) – tato varianta není zrovna nejšťastnější ani nejrychlejší.

- Provést backup a následný restore celé databáze, protože při obnově databáze ze zálohy se obnovují pouze aktuální data – tato varianta má velkou nevýhodu v tom, že databáze je po dobu backup/restore procesu nedostupná třeba i na několik hodin. Výhoda varianty spočívá v tom, že backup/restore přepočítá indexy, smaže stará data a celkově odlehčí databázovému stroji při vyhledávání v datech.
- Manuální spuštění tzv. Total Garbage Collection s parametrem sweep.

Total Garbage Collection slouží pro manuální spuštění GC na všech datech v databázi. Jedná se o proces běžící v separovaném vlákně (stejně jako u Background GC), který čte data ze všech tabulek databáze. Tento proces běží vždy ve variantě Cooperative GC, kdy data ihned kontroluje a odmazává. Total Garbage Collection (někdy také označovaný jako Sweep) lze mimo manuálního spouštění také nastavit, aby se spouštěl pravidelně dle předem stanovených podmínek (default nastaven na 20000). Příkaz pro změnu default hodnoty:

```
gfix -house 20000
```

Hodnota parametru neznamená, že se GC spouští každých 20000ms (nebo s), jedná se o počet transakcí mezi hodnotou OIT (Oldest Interesting Transaction) a next transaction number. Obě hodnoty si databázový stroj uchovává v paměti, kdy OIT znamená nejstarší číslo transakce, která je ještě pro GC zajímavá (tzn. po této transakci následuje nejstarší nepotvrzená transakce OST), next transaction number je číslo transakce, které bude přiděleno následující nově vytvořené transakci.

3.6.3 Časté výkonnostní problémy s databází

Mezi časté výkonnostní problémy s databází Firebird patří rychlost odezvy a vyhledávání dotazů. Pokud se výkon databáze z neznámého důvodu výrazně sníží, příčinou může být právě velké množství Data page záznamů. Tuto situaci můžeme zkontrolovat pomocí čísel OST a next transaction number. Pokud rozdíl těchto čísel je větší než předpokládaný počet spojení všech uživatelů (počet uživatelů * počet možných spojení na databázi), tak může být příčinou výkonnostních problémů transakce, která nebyla správně ukončena a je stále aktivní. Nejednoduší metodu pro získání těchto informací poskytne program gstat, který je součástí standardní instalace Firebird databáze.

```
gstat -h /cesta/k/databazi
```

Výsledkem předchozího příkladu může být následující výpis:

```
Database "/cesta/k/databazi.fdb"
Database header page information:
      Flags                   0
      Checksum                12345
      Generation              4863663
      Page size               4096
      ODS version             11.2
      Oldest transaction      3040352
      Oldest active           3040353
      Oldest snapshot         3040353
      Next transaction        3042221
      Bumped transaction      1
      Sequence number         0
      Next attachment ID      1821410
      Implementation ID       24
      Shadow count            0
      Page buffers            0
      Next header page        0
      Database dialect        3
      Creation date           Jan 11, 2016 0:08:44
      Attributes              force write
Variable header data:
      Sweep interval:         20000
      *END*
```

Z výpisu lze mimo jiných informací vyčíst OIT 3040352, číslo poslední aktivní transakce OST 3040353, číslo next transaction 3042221 nebo počet transakcí ke spouštění Total GC (20000). Tato situace na výpisu je pro danou databázi standardní a databáze nevykazuje žádné výkonnostní problémy (počet evidovaných verzí transakcí je cca 2000). V tomto případě, (jedná se o statistiku hlavní firemní databáze firmy SOMA eng.) pokud rozdíl OIT a next transaction je větší jak cca 10 000, se databáze začne výrazně zpomalovat. Podle čísla OST lze dohledat v systémových tabulkách, která transakce (který uživatel) blokuje GC a zpomaluje celý systém. Tím lze snadněji vyhledávat problémové části programu, které dlouho drží aktivní transakce a zpomaluje všechny ostatní uživatele. Program gstat umožňuje zobrazovat mnohem více informací, které mohou být užitečné pro vyhledávání výkonnostních

problémů s databází (velikost Data page, velikost jednotlivých tabulek, počet Data page záznamů, …).

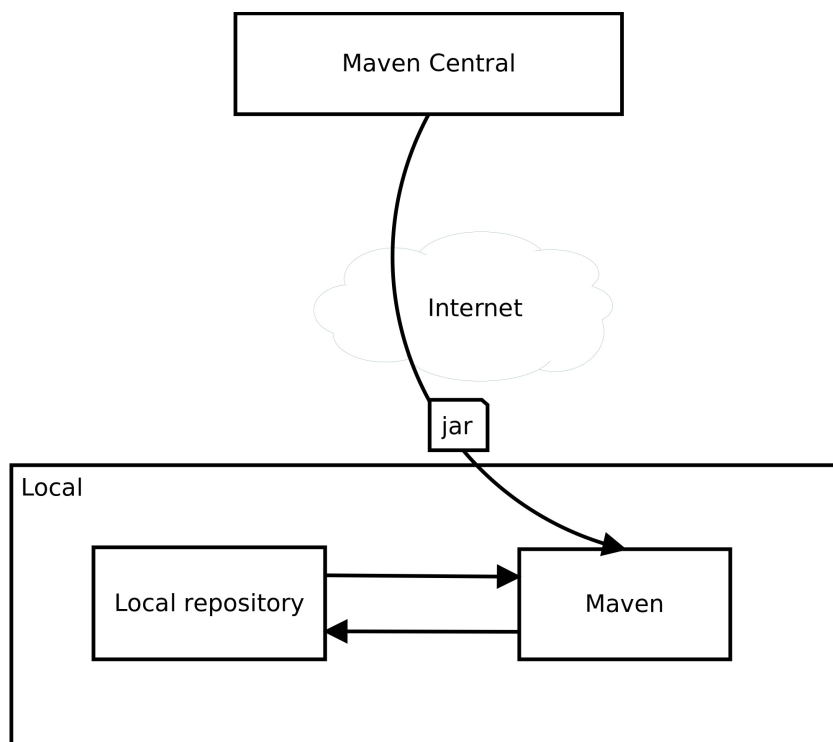
3.7 Apache Maven

Ačkoli technologie Apache Maven (dále jen Maven) není přímo zasahující do výsledného kódu webového portálu, je velice důležitou částí při samotném vývoji. Jedná se o technologii pro projektový management, zahrnující zjednodušení buildů aplikací, testování, reportování a balíčkování (packaging). První verze projektu Apache Maven byla vydána v roce 2004, nicméně historie projektu sahá již do roku 2000, kdy základy projektu vznikaly jakou součást Apache Jakarta Alexandria project. Projekt je veden jako open-source a postupně se dostal do čela mezi nástroji svého druhu. Poslední doporučovanou stabilní verzi začátkem roku 2016 byla verze 3.3.9. [MAVEN]

Na začátku vývoje (nejen) webových aplikací se hodně času stráví tzv. projektovým layoutem – návrh a struktura jednotlivých složek, uspořádání tříd a konfiguračních souborů. Tento layout adresářové a souborové struktury může být napříč různými vývojovými týmy nebo jednotlivými projekty velmi odlišný. Z tohoto důvodu může být vstup nového vývojáře nebo přestup vývojáře mezi projekty velmi náročný na orientaci v takovýchto projektech. Tento problém při vývoji jednotlivých projektů lze řešit právě pomocí technologie Apache Maven. Maven umožňuje sjednotit adresářovou strukturu a organizaci v projektu. Maven poskytuje doporučení, kde mají být uloženy jednotlivé součásti projektu jako je zdrojový kód, testovací kód nebo konfigurační soubory. Příkladem může být zdrojový Java kód, který by měl být uložen v adresářové struktuře src/main/java. Takto určené rozdělení zlepšuje navigaci a usnadňuje pochopení každého projektu spravovaného pomocí technologie Maven. [VARANASI]

Mezi výhody použití technologie Maven, které jsou na první pohled přínosné při počáteční konfiguraci existujícího projektu (například novým členem týmu) je správa závislostí. Každý projekt používající Maven obsahuje soubor pom.xml, který obsahuje definice použitých knihoven a projektů s přesným označením verze, která je při vývoji použita. Maven z tohoto souboru již sám samotné knihovny (potřebné pro běh aplikace) stahuje z centrálního repositáře a zkracuje tak významně čas, který by byl potřeba pro manuální hledání správných verzí knihoven a import těchto knihoven do projektu. Soubor pom.xml tedy poskytuje odkazy na potřebné knihovny s přesně použitou verzí a na použité dokumentační nástroje. Projekt Maven může být rozšířen o spoustu různých doplňků (tzv. plug-in), kterých v současnosti existuje několik stovek. Tato rozšíření mohou významně zautomatizovat různé pracovní postupy při vývoji (například pravidelný build, dokumentování nebo dotazování na různé verze projektu). Na obrázku 14 je znázorněn pohled na dependency management projektu Maven, kde je naznačen centrální repositář, odkud se stahují potřebné knihovny do lokálního

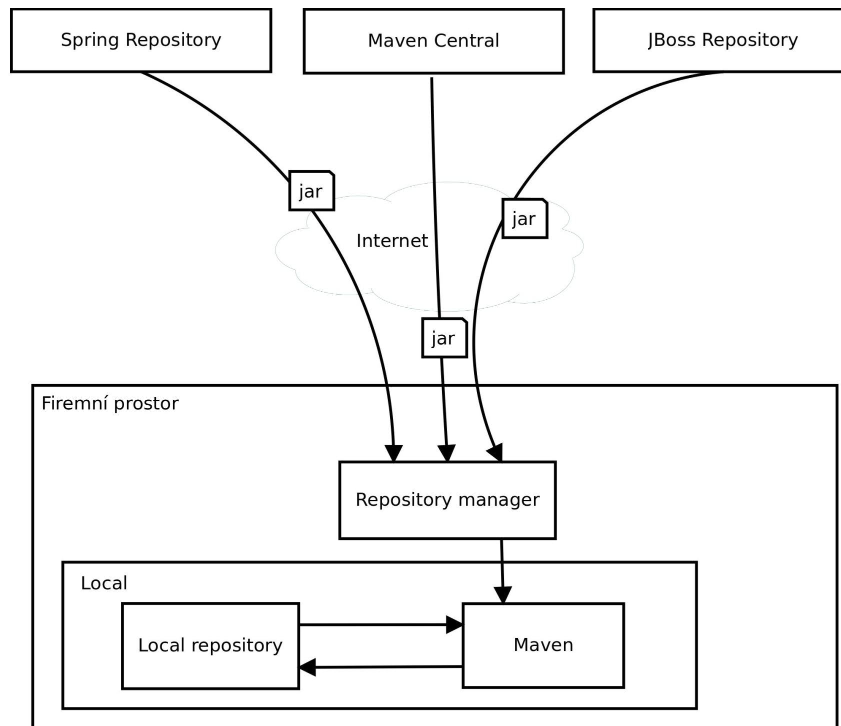
repositáře každého uživatele. Centrální repositáře jsou obhospodařovávány zpravidla třetími stranami, nicméně mezi výchozí repositáře patří repo.maven.apache.org a uk.maven.org.



Obrázek 14: Maven dependency management. [VARANASI]

U této varianty nastavení může ve větších firmách s více vývojáři nastat několik problémů. Prvním problémem může být sdílení vlastních knihoven, kdy z bezpečnostních nebo licenčních důvodů není možné firemní knihovny nahrávat do veřejného centrálního repositáře, a to i přesto, že je potřeba knihovny sdílet mezi jednotlivými vývojovými týmy. Ve vývojové firmě může nastat situace, kdy je projekt vyvíjen pouze pomocí předem zvolených knihoven (například z důvodu licencování), nebo za výběr potřebných knihoven zodpovídá jeden člověk. Pokud však má každý vývojář přístup k centrálnímu repositáři, může si stahovat a nastavovat knihovny dle jeho vlastního uvážení.

Předchozí popsané problémy lze řešit pomocí centrálního firemního repositáře, kdy jednotliví vývojoví pracovníci si do svého lokálního Maven repositáře nastavují knihovny z tohoto firemního repositáře. O tento firemní repositář se pak stará zodpovědná osoba, která určuje jednotlivé verze a knihovny pro vývoj jednotlivých projektů. Obrázek 15 tuto situaci centrálního firemního repositáře naznačuje graficky, navíc jsou naznačeny i jiné centrální repositáře třetích stran (Spring, Jboss). Mezi další výhody firemního repositáře může patřit rychlost a vytíženost internetové linky při stahování většího množství knihoven více vývojáři najednou, zvláště pak z repositářů obsahujících známé a často používané knihovny (např. Spring MVC, apod.), ze kterých stahují data vývojáři z celého světa.



Obrázek 15: Enterprise Maven repository architecture. [VARANASI]

Ovládání technologie Maven lze provádět pomocí příkazového řádku, nicméně většina dnešních vývojových IDE nástrojů plně Maven podporuje a není tedy nutná ruční konfigurace. Technologie Maven také obsahují tzv. archetypy. Jedná se vlastně vzorové adresářové struktury obsahující předem potřebné soubory ke zvolené technologií. Příkladem může být archetyp pro Spring MVC projekt, který usnadní vytvoření základní adresářové struktury včetně potřebných souborů (web.xml, pom.xml, dispatcher-servlet, aplication-servlet apod.).

3.7.1 Alternativy Maven

Mezi alternativy k použití technologie Maven z pohledu funkčnosti patří technologie Ant (v kombinaci s Ivy) a Gradle. Technologie Ant podobně jako Maven slouží k automatizaci buildů, nicméně sama o sobě neobsahuje automatickou správu závislostí (dependency management), k tomu slouží rozšíření Ivy. Ant+Ivy poskytují možnost konfigurace závislostí (knihoven) a spolu tak mohou v určitých situacích konkurovat technologií Maven.

Technologie Gradle je jazyk pro automatizaci. Nejčastěji se asi bude jednat o automatizaci buildů. Nástroj Gradle však není nijak omezen a zautomatizovat si lze třeba i zasílání měsíčních výkazů. Pro popis toho, co je potřeba zautomatizovat se používá DSL (Domain Specific Language). Výchozí těžiště Gradlu leží v buildování. V tomto směru na něj lze nahlížet jako na nástroj další generace v linií Ant → Maven → Gradle, přičemž ze svých

předchůdců si bere to nejlepší: z Antu jeho sílu a flexibilutu, z Mavenu konvence, dependency management a pluginy. Právě pluginy jsou důležitou součástí při používání Gradlu. Jádro Gradlu toho totiž samo o sobě moc neumí ale spolu s dalšími pluginy se stává mocným nástrojem. [KOTACKA]

4. Popis použitých metod a technického řešení portálu

Webový portál je vyvíjen v prostředí Netbeans pomocí technologií a frameworků popsaných v předchozích kapitolách části 3 Popis použitých technologií. Autor této práce je zároveň autorem celé back-endové části projektu kooperačního portálu. Front-end část zpracoval externí dodavatel, který s firmou SOMA dlouhodobě spolupracuje v oblasti marketingu a webového designu. Tato práce se proto touto částí nezabývá a dále tyto použité technologie nebudou popisovány (HTML, CSS ani JavaScript).

4.1 Požadované případy užití portálu

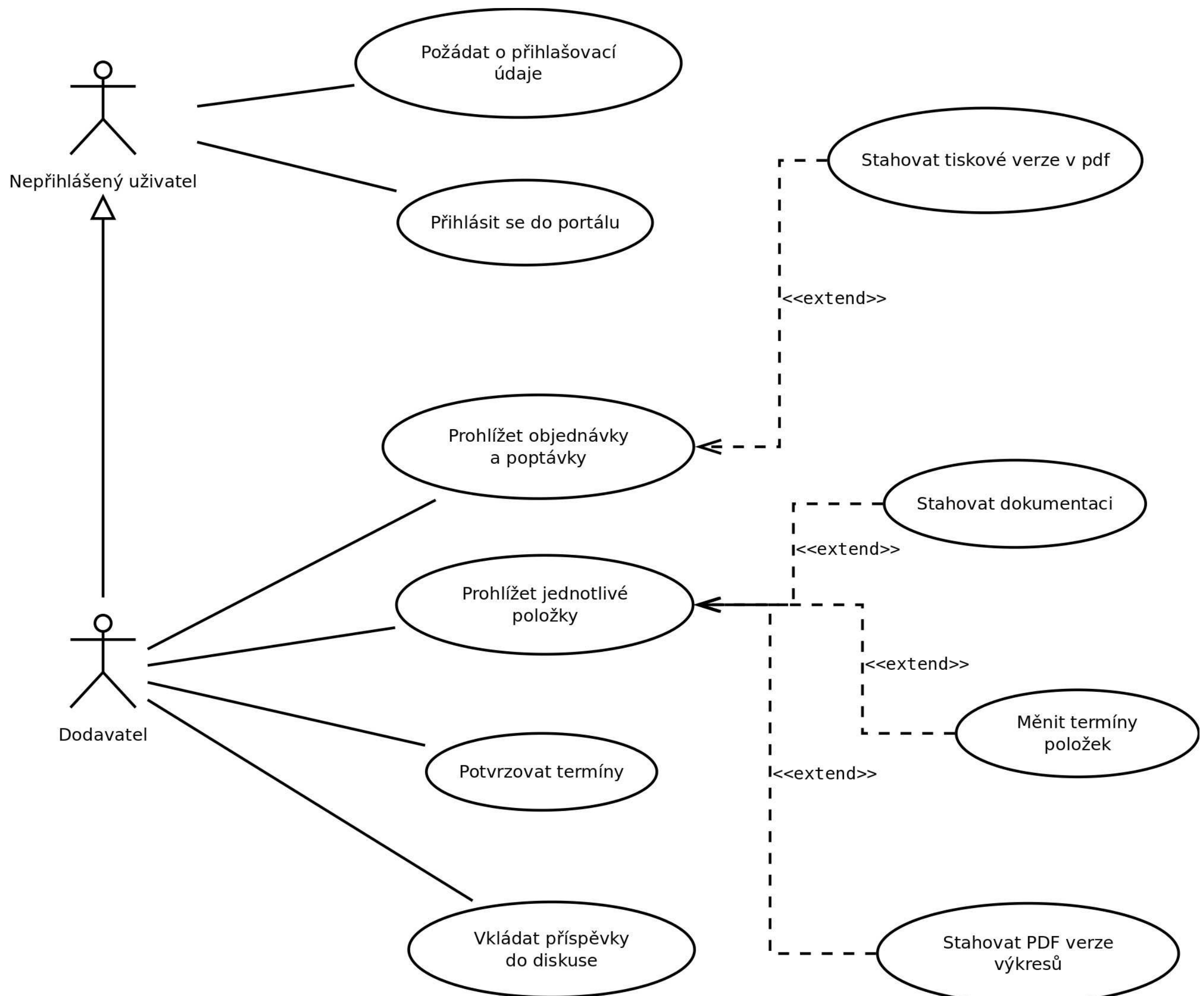
Před samotnou implementací portálu bylo nutné se zodpovědnými osobami sjednotit požadavky na portál, aby bylo předem určené, co bude výsledkem celého projektu a do jakých oblastí bude portál nově zasahovat. Předem bylo určeno, že portál bude sloužit pro dodavatele firmy SOMA, kteří budou moci pracovat s poptávkami a objednávkami ze strany SOMA. Portál se tedy týká oddělení kooperace (objednávky služeb a operací na vyráběných dílech) a oddělení nákupu, kde se nakupuje materiál pro výrobu.

Důležité funkční požadavky jsou zobrazeny v use-case diagramu na obrázku 16. Následující část podrobněji popisuje jednotlivé případy užití.

- Požádat o přihlašovací údaje – uživatel má možnost zobrazit úvodní stránku s odkazem na instrukce pro přidělení přihlašovacích údajů.
- Přihlásit se do portálu – nepřihlášený uživatel portálu má možnost se přihlásit. Bez přihlášení není možnost přistupovat k dalším částem portálu.
- Prohlížet poptávky a objednávky – přihlášený uživatel má možnost přehledně zobrazovat jednotlivé poptávky a objednávky. Přehledným způsobem jsou mu zobrazeny blížící se termíny a nově přidané poptávky a objednávky ze strany SOMA.
    - Stahovat tiskové verze v Pdf – uživatel má možnost si stahovat tiskové verze poptávek a objednávek v takové formě, ve které byly poptávky / objednávky zasílány před nasazením webového portálu.
- Prohlížet jednotlivé položky – uživatel má možnost prohlížet detaily jednotlivých poptávek a objednávek. U každé položky musí být dostatek informací okolo požadované služby nebo materiálu, včetně požadovaných termínů dodání, množství, identifikačních údajů a dalších důležitých informací pro přesnou identifikaci požadavku (popis operací, katalogová čísla apod.).

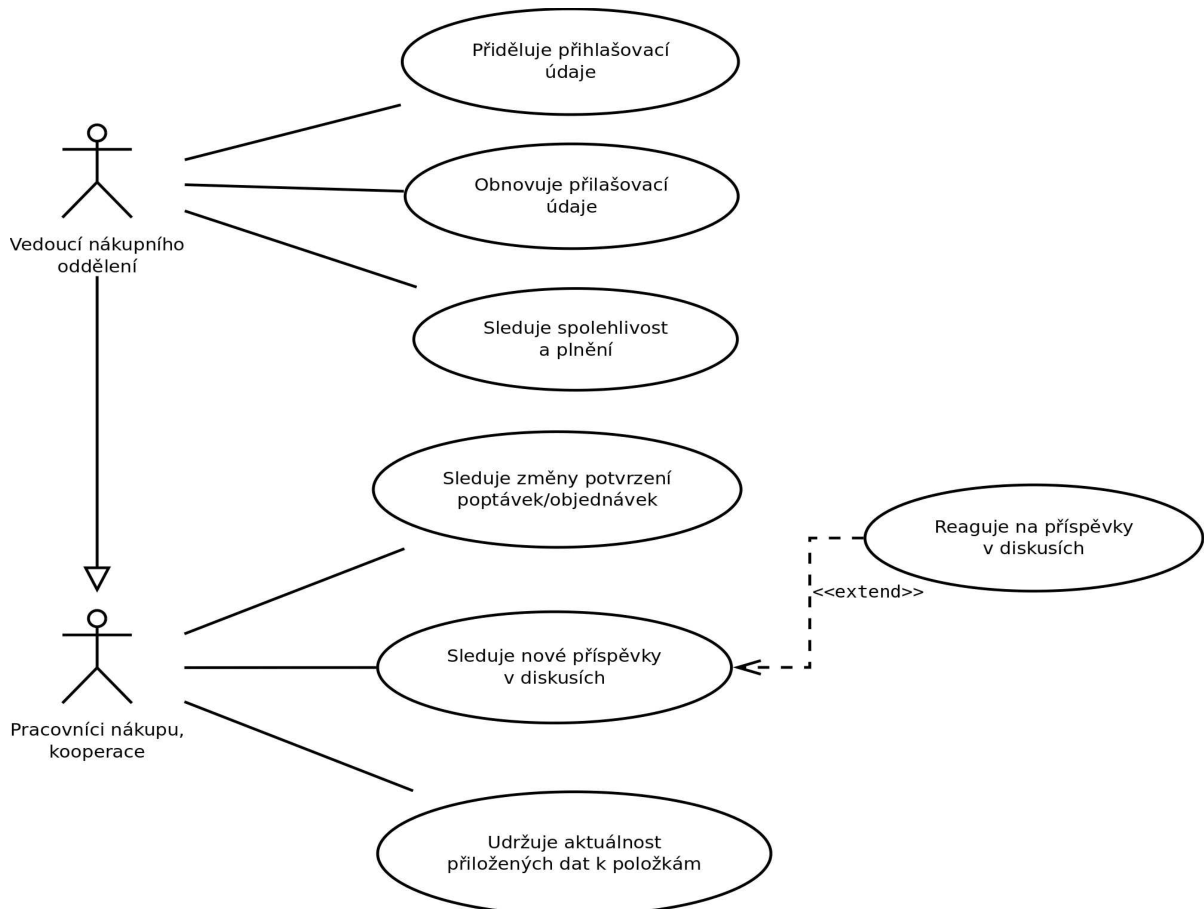
- Stahovat dokumentaci – dodavatel má možnost k jednotlivým položkám objednávek stahovat dokumentaci, např. postupky průvodky, 3D a 2D data k CNC výkresům a jiné dostupné dokumenty k položkám.
- Měnit termíny položek – dodavatel má možnost před potvrzením poptávky (nebo objednávky) u jednotlivých položek měnit jejich termíny v závislosti na jeho výrobních nebo skladových kapacitách.
- Stahovat Pdf verze výkresů – dodavatel má možnost si stahovat Pdf verze výkresů k jednotlivým položkám objednávky.

- Potvrzovat termíny – uživatel má možnost jednoduchou akcí potvrdit celou poptávku nebo objednávku a tím vyjádřit svůj souhlas s požadavky firmy SOMA.
- Vkládat příspěvky do diskuse – uživatel může přidávat ke každé jednotlivé poptávce nebo objednávce příspěvky do diskuse. Tím může vyjádřit svůj nesouhlas s objednávkou, argumentovat požadované podmínky dodání nebo jen vkládat jiné upřesňující dotazy týkající se poptávky nebo objednávky.



Obrázek 16: Diagram užití webového portálu dodavatelem. [autor]

Výše popsané případy užití jsou přímo navázané na webový portál. Z pohledu pracovníků firmy SOMA je však nutné rozšířit stávající vnitřní ERP systém o nové funkce vztahující se právě k webovému portálu. Pracovníci firmy SOMA obsluhují poptávky a objednávky i nadále přes ERP systém. Nově požadované případy užití ERP systému jsou znázorněny v diagramu na obrázku 17.



Obrázek 17: Diagram nových funkčních rozšíření stávajícího ERP systému. [autor]

Mezi nově implementované funkční požadavky ERP systému pro vedoucího nákupního oddělení patří:

- Přiděluje přihlašovací údaje – na základě vlastního rozhodnutí (nebo žádosti dodavatele) určuje firmy, které budou mít přístup k webovému portálu. Přihlašovací údaje systém generuje automaticky pomocí tlačítka v části pro správu firem (agenda Firmy).
- Obnovuje přihlašovací údaje – vedoucí oddělení má možnost měnit hesla nebo přímo odebírat přihlašovací údaje dodavatelů.
- Sleduje spolehlivost plnění – pro snazší přehled o plnění a spolehlivosti jednotlivých firem je v systému vytvořen nový nástroj pro různá vyhodnocování firem. Nově se také každý měsíc vybraným firmám odesílá dopis o plnění a spolehlivosti jejich dodávek.

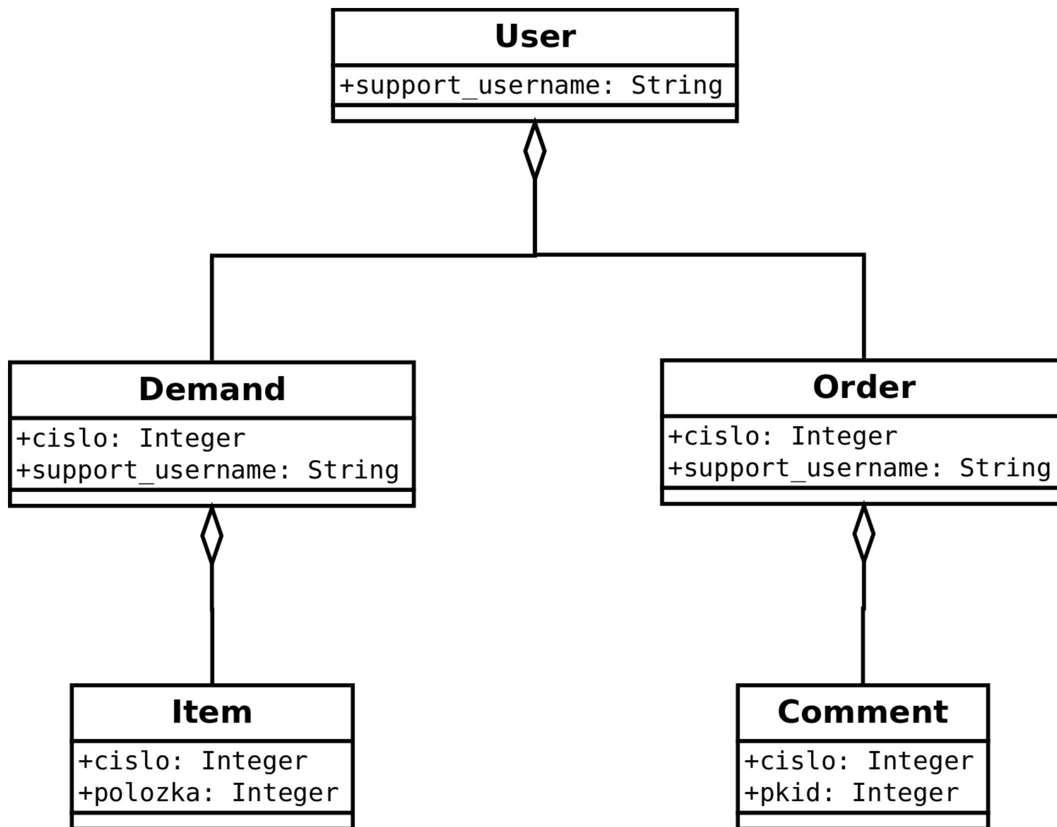
Pro jednotlivé pracovníky firmy, kteří objednávají nějaké služby nebo materiál u dodavatelů jsou v systému přidány následující rozšíření:

- Sleduje změny potvrzení poptávek / objednávek – pracovník má možnost si zobrazit přehled změn na jeho objednávkách ze strany dodavatele přes webový portál. Pokud tedy uživatel portálu provede nějakou změnu, příslušný pracovník SOMA má možnost se o změně dozvědět. Změny z webového portálu také chodí jako upozornění na mail.
- Sleduje nové příspěvky v diskusích – stejně jako jiné změny, tak i změny v diskusích k jednotlivým poptávkám a objednávkám jsou doručeny do mailové schránky příslušného pracovníka. Jednotlivé diskuse je možné sledovat v detailech u každé poptávky nebo objednávky.
  - Reaguje na příspěvky v diskusích – pracovník SOMA má nově možnost spravovat příspěvky k jeho objednávkám přímo v systému (vkládat / mazat / měnit příspěvky).
- Udržuje aktuálnost přiložených dat k položkám – zejména pracovníci kooperace jsou upozorňováni na neaktuální data při tvorbě poptávek a objednávek, která si může uživatel portálu stáhnout. Pracovníci kooperace tak musí u takových dat zajistit jejich aktuálnost s ostatními pracovníky příslušných oddělení (technologie, konstrukce).

4.2 Datový model webového portálu

Webový portál získává data ze stávající firemní databáze Firebird. Webový portál pro přístup k datům používá ORM technologií Hibernate (viz kapitola 3.5). Hibernate technologie mapuje data z databáze na objekty. Základní datový model je znázorněn na obrázku 18. Na obrázku jsou zobrazeny pouze klíčové atributy (z databázového pohledu se jedná o primární a cizí klíče).

Pro usnadnění mapování jsou v databázi vytvořeny pohledy, které jsou pojmenovány stejně jako třídy (pouze s prefixem KOO_). Databáze tedy obsahuje pohledy KOO_USERS, KOO_DEMANDS, KOO_ORDERS, KOO_ITEMS a KOO_COMMENTS. Tyto pohledy jsou sestaveny z jednotlivých tabulek, tak aby zobrazovaly všechny požadovaná data.



Obrázek 18: Model datových tříd portálu. [autor]

Například třída Item obsahuje všechny potřebné informace o každé jednotlivé položce objednávky, které webový portál může potřebovat pro zobrazení nebo další zpracování. V následující ukázce je zobrazena část zdrojového kódu této třídy.

```
@Entity
@Table(name = "KOO_ITEMS")
public class Item implements Serializable {
    @Id
    @Column(name = "OBJ_POL")
    private String objPol;

    @Column(name = "OBJEDNAVKA")
    private int cislo;

    @Column(name = "POLOZKA", columnDefinition = "smallint")
    private int polozka;
...
```

V databázi jsou tyto informace uloženy celkem v 9 tabulkách, které jsou sjednoceny do jednoho pohledu KOO_ITEMS. Mapování pak probíhá jednoduše pomocí anotace u každého atributu třídy, jak je patrné z ukázky.

Vytvoření pohledu KOO_ITEMS a mapování třídy na pohled má několik výhod. Hlavní výhodou je jednoduché anotování třídy na pohled (anotace @Table(name = …)). Druhou výhodou je, že uživatel webového portálu má přístup do databáze pouze pro pohled KOO_ITEMS a k příslušným tabulkám už oprávnění nemá (nedostane se tak k informacím, které pro něj nejsou určené). Avšak největším přínosem je sjednocení rozdílného uložení materiálových položek (viz Obrázek 3) a položek kooperačních objednávek (viz Obrázek 4). Definice pohledu je naznačena v následující ukázce, pro lepší čitelnost SQL dotazu jsou některé části vynechány a nahrazeny třemi tečkami.

```
create or alter view KOO_ITEMS as
  select O.OBJEDNAVKA || OP.POLOZKA OBJ_POL, O.OBJEDNAVKA, OP.POLOZKA,
      OP.NAZEV_MATERIALU NAZEV, MM.VYKRES_DILU VYKRES,
      0 ZAKAZKA, 0 VYROBNI_CISLO, OP.MNOZSTVI, OP.JEDNOTKA,
...
      coalesce(ZARV.ARCHIV_VYKRES, ZARV2.ARCHIV_VYKRES) ARCHIV_VYKRES,
      ZARV.DATA_DXF, ZARV.DATA_IGES, ZARV.DATA_STEP
    from MTZ_OBJEDNAVKY O
      left outer join ODB_FIRMY_PRISTUPY FP on ...
      join MTZ_OBJEDNAVKY_POLOZKY OP on ...
      left outer join MTZ_MATERIAL MM on ...
      left outer join I2_GET_ID_N(MM.ID_NAKUPOVANY_DIL) IDN on ...
      left outer join ZAR_VYKRESY ZARV on ...
      left outer join ZAR_VYKRESY ZARV2 on ...
  union all
  select O.OBJEDNAVKA || VP.POLOZKA_OBJEDNAVKY OBJ_POL, O.OBJEDNAVKA,
      VP.POLOZKA_OBJEDNAVKY POLOZKA, coalesce (ZARV.NAZEV, P.VYKRES)
      ...
      iif (max (VP.ODVEDENO) = '*', max(VP.MNOZSTVI), 0),
      max (VP.TERMIN_ZHOTOVENI) TERMIN_ZHOTOVENI,
      max (VP.DATUM_EXPEDICE_MATERIALU) DATUM_EXPEDICE_MATERIALU,
...
    from MTZ_OBJEDNAVKY O
      left outer join ODB_FIRMY_PRISTUPY FP on FP.ICO = O.ICO
```

```
      join ZAK_VYROBNI_PRIKAZY VP on VP.OBJEDNAVKA = O.OBJEDNAVKA
      join ZAK_POSTUPY P on ...
      left outer join ZAR_VYKRESY ZARV on ...
      left outer join ZAR_VYKRESY ZARV2 on ...
      left outer join I2_GET_ID_V(ZARV.ID_VYRABENY_DIL, 0) IDV on 1=1
      group by O.OBJEDNAVKA, VP.POLOZKA_OBJEDNAVKY, …
```

Z ukázky je patrné, že v pohledu je částečně také zahrnuta určitá logika sjednocení dvou odlišných způsobů uložení položek materiálových a kooperačních objednávek. Při vynechání pohledu a mapování objektu přímo pomocí Hibernate anotací na jednotlivé tabulky a sloupce by došlo ke zbytečnému zesložitění celého modelu.

4.3 Zabezpečení aplikace

Webový portál, jak již bylo řečeno, je v základu postaven na webovém frameworku Spring MVC, proto se jako nanejvýš vhodná varianta zabezpečení nabízí bezpečnostní framework Spring Security. V této kapitole jsou popsány konkrétní nastavení tohoto frameworku ve webovém portálu. Obecný popis frameworku je v kapitole 3.4 Spring Security framework.

Důležitým souborem celého frameworku je soubor spring-security.xml v adresáři WEB-INF. Cesta k tomuto souboru je nastavena ve web.xml.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>...,/WEB-INF/spring-security.xml</param-value>
</context-param>
```

Samotný spring-security.xml soubor obsahuje dva důležité konfigurační elementy <html> a <authentication-manager>. Následující ukázky kódu zobrazují element <html> včetně některých jeho podřízených elementů.

```
<http auto-config="true" use-expressions="true">
```

Auto-config zajistí výchozí nastavení všech jinak nenastavených parametrů, atribut use-expressions povolí používání výrazů pro definici přístupů (výraz např. hasRole(SKLADNIK) je převeden na boolean hodnotu, podle toho, zda aktuální uživatel má přidělenou roli SKLADNIK.

```
      <intercept-url pattern="/**" requires-channel="https"/>
```

Tento element <intercept-url ...> nastavuje, že komunikace mezi prohlížečem a serverem bude probíhat přes protokol https a to na všech stránkách. V případě potřeby nezabezpečeného protokolu na vybraných stránkách se tento element vkládá vícekrát s různými atributy pattern.

```
<access-denied-handler error-page="/403.html"/>
<intercept-url pattern="/403*"  access="permitAll"/>
```

Element access-denied-handler nastaví výchozí mapování pro nepovolený přístup. Tato stránka je pak zobrazena vždy, když se uživatel pokusí vstoupit na stránku, ke které nemá oprávnění. Stránka se také může zobrazit přihlášenému uživateli, který nemá platný CRLF token. Element <intercept-url> nastavuje přístup pro všechny uživatele právě na stránku 403.html s informací o nepovoleném přístupu.

```
<intercept-url pattern="/index*" access="permitAll"/>
<intercept-url pattern="/contact-logged*"
                 access="isAuthenticated()"/>
<intercept-url pattern="/contact*"  access="isAnonymous()"/>
<intercept-url pattern="/resources/**" access="permitAll"/>
<intercept-url pattern="/**" access="isAuthenticated()"/>
```

Sled elementů <intercept-url> postupně nastavuje povolení na vstupní stránku (index.html) pro všechny uživatele. Dále je zajímavé zmínit stránky contact-logged a contact. Obě tyto stránky zobrazují kontaktní informace, nicméně na stránku contact-logged.html vstupují pouze přihlášení uživatelé (isAuthenticated), naopak na stránku contact.html pouze nepřihlášení uživatelé (isAnonymous). Důležité je také povolit přístup k CSS a jiným zdrojovým souborům, potřebným pro správné fungování a zobrazení úvodní stránky (povolení přístupu do adresáře resources, kde se tyto soubory nacházejí). Poslední element z předchozí ukázky určuje, že na všechny ostatní stránky, které nebyly výslovně povoleny v předchozích případech, je požadováno přihlášení (isAuthenticated).

```
<form-login
      login-processing-url="/j_spring_security_check"
      login-page="/index.html"
      default-target-url="/dashboard.html"
      authentication-failure-handler-ref="customFailureHandler"
      username-parameter="j_username"
      password-parameter="j_password" />
```

V elementu form-login se nastavují stránky, kde se nachází možnost přihlášení uživatele (login-page), výchozí stránka po přihlášení (default-target-url) a reference na vlastní obsluhu neúspěšných přihlášení (authentication-failure-handler-ref).

```
        <logout logout-success-url="/index.html?logout"
           logout-url="/logout"
           delete-cookies="JSESSIONID"
           invalidate-session="true" />
```

Část s elementem <logout> určuje přesměrování v případě odhlášení uživatele a také to, že se má smazat JSESSIONID a zneplatnit celé sezení (session).

```
        <csrf />
</http>
```

Poslední část popisu elementu <http> obsahuje pouze povolení ochrany CSRF. Ve výchozím nastavení ve verzi Spring Security 4.0 je tato ochrana povolena automaticky, zde je to uvedeno z důvodů, aby při čtení konfiguračního souboru bylo hned jasné, že CSRF je povoleno.

Další část z konfiguračního souboru se týká načítání autentizačních údajů z databáze. Jako poskytovatel těchto údajů je zde jdbc-user-service. V něm je přesně specifikovaný SQL dotaz pro ověření uživatelského jména a hesla a také dotaz pro načtení uživatelských rolí (v ukázce pouze naznačeno). Pokud jsou hesla v databázi uložena v hash formátu, musí se tato volba nastavit v elementu <password-encoder>.

```
<authentication-manager>
     <authentication-provider>
          <password-encoder hash="bcrypt" />
          <jdbc-user-service data-source-ref="dataSource"
               users-by-username-query=
               "select USER_NAME, PASSWORD, ENABLED
                    from ... where USERNAME = ? "
               authorities-by-username-query=
               "select USER_NAME, ROLE
                    from ... where USERNAME = ? "/>
     </authentication-provider>
</authentication-manager>
```

Poslední část konfiguračního souboru spring-security.xml obsahuje vlastní obsluhu neúspěšných přihlášení. Na tuto beanu jsme odkazovali v elementu <form-login>. V ukázce je nastaveno přesměrování na stránku index.html s parametrem error=1 v případě špatně zadaných přihlašovacích údajů. Jako výchozí stránka při ostatních výjimkách se volá index.html s parametrem nodatabase. O zobrazení konkrétní zprávy o neúspěšném přihlášení se stará přímo stránka index dle předaného parametru (error nebo nodatabase).

```
<beans:bean id="customFailureHandler"
class="org.springframework.security.web.
      authentication.ExceptionMappingAuthenticationFailureHandler">
      <beans:property name="exceptionMappings">
            <beans:props>
                  <beans:prop key="org.springframework.security.
                              authentication.BadCredentialsException">
                        /index.html?error=1
                  </beans:prop>
            </beans:props>
      </beans:property>
      <beans:property name="defaultFailureUrl" value="/index.html?
nodatabase" />
  </beans:bean>
```

Na stránce index.html můžeme přímo přistupovat k textu výjimky pomocí následujícího EL výrazu. Tím získáme možnost zobrazit celé chybové hlášení, které může být při hledání důvodů neúspěšného přihlášení velmi užitečné, zejména ve fázi vývoje aplikace.

```
${sessionScope["SPRING_SECURITY_LAST_EXCEPTION"].message}
```

4.4 Mapování zdrojů

Ve webovém portálu je implementováno hned několik kontrolerů, které se starají o správné mapování požadavků. Kontrolery jsou vytvořeny jako Java třídy s patřičnou anotací ze Springu (@Controller). V kontrolerech je využita vlastnost Spring MVC frameworku Dependenci injection (anotace @Autowired). Injektují se tak zejména třídy, které slouží kontrolerům k práci s ORM frameworkem Hibernate.

Každý kontroler má pokrytou jednu funkční oblast poskytovaných funkcí:

- MainController.java – hlavní kontroler, který slouží pro mapování jednotlivých stránek celého webového portálu,
- OperativeController.java – tento kontroler zajišťuje obsluhu požadavků, které budou měnit data v databázi (potvrzení objednávky nebo změny termínů),
- CommentController.java – slouží pro obsluhu požadavků, které se týkají práce s komentáři k jednotlivým objednávkám,
- PdfController.java – tento kontroler umožňuje obsluhovat požadavky ke stažení různých tiskových sestav, výkresů a dat k výkresům.

4.4.1 Stahování dat a tiskových sestav

V předchozí kapitole je zmíněný PdfController pro stahování dat. Současný informační systém umožňuje zaměstnancům tisknout tiskové sestavy, které jsou vytvořeny v aplikaci iReport pomocí knihoven JasperReport. Třídy a metody v informačním systému jsou již vytvořeny a úspěšně otestovány, proto webový portál tyto třídy používá. Jednotlivé knihovny jsou do portálu přidány přes lokální Maven repositář. Tiskové sestavy jsou pak přímo generovány z databáze, protože databáze obsahuje jak samotnou definici každé tiskové sestavy, tak také data potřebná k naplnění tisku.

PdfController poskytuje také možnost stahovat výkresy a data k výkresům. Tyto data nejsou (na rozdíl od tiskových sestav) uložena přímo v databázi, ale jsou uložena na zvláštním podnikovém serveru. Přístup k tomuto serveru není pro server, kde běží webový portál, povolen přímo, ale je nutné z bezpečnostních důvodů data stahovat pomocí tzv. CGI scriptů. Tyto scripty umožňují pomocí url dotazů poskytovat data z jinak nepřístupného serveru. Touto metodou se řeší nejen přístup k datům z webového portálu, ale také ze stávajícího informačního systému. Podrobnější informace jsou uvedeny v kapitole 5.2.1 Spolupráce mezi WWW serverem a CGI skriptem.

4.5 Přístup k datům

Webový portál potřebuje přistupovat k firemní databázi. O správné mapování objektů (viz Obrázek 18) na pohledy z databáze se stará technologie Hibernate. Konfigurace je uložena v souboru hibernate.cfg.xml. Následuje popis některých nastavení tohoto konfiguračního souboru.

```
<hibernate-configuration>
  <session-factory>
     <property name="dialect">
           org.hibernate.dialect.FirebirdDialect
     </property>
```

Nastavení firebird dialektu.

```
     <property name="current_session_context_class">thread</property>
```

Tato volba nastavuje, že session bude ukončena po každé transakci. Teoreticky se tak nemusí uzavírat ručně.

```
     <property name="cache.provider_class">
           org.hibernate.cache.NoCacheProvider
     </property>
```

Volba cache.provider_class zamezuje použití second-level cache.

```
      <property name="show_sql">true</property>
```

Tato volba vypisuje do standardního výstupu každý SQL dotaz, který Hibernate generuje, to může být vhodné zejména při vývoji aplikace. Lze tak sledovat jak přesně, jak často a jaké SQL dotazy Hibernate generuje do databáze.

```
      <property name="hbm2ddl.auto">validate</property>
```

Hibernate automaticky validuje nebo exportuje DDL schéma databáze při vytváření session. Volba hbm2ddl.auto validate zajistí, že Hibernate bude schéma pouze validovat, ale nebude provádět žádné změny schématu.

Poslední část konfiguračního souboru definuje modelové třídy, které Hibernate bude mapovat na pohledy.

```
      <mapping class="cz.soma.portal.kooperace.model.Demand" />
      <mapping class="cz.soma.portal.kooperace.model.Item" />
      <mapping class="cz.soma.portal.kooperace.model.Order" />
      <mapping class="cz.soma.portal.kooperace.model.Comment" />
      <mapping class="cz.soma.portal.kooperace.model.User" />
  </session-factory>
</hibernate-configuration>
```

4.5.1 Třídy pro přístup k datům

Každá jednotlivá modelová třída webového portálu má svojí třídu, přes kterou přistupuje k datům. Příkladem může být třída komentáře Commnet.java. Pro práci s touto třídou a synchronizaci s databází se využívá rozhraní CommnetDao.java.

Toto rozhraní obsahuje metody používané v portálu pro získání nebo uložení komentářů.

```
public interface CommentDao {
    List<Comment> getComment(String id, int cislo, String typ);
    boolean saveComment(Comment comment);
}
```

Jako implementační třída je použita třída CommentDaoImpl. Tato třída dědí z rozhraní CommentDao obě metody (getComment i saveComment). Pro správné vytváření session potřebuje třída CommentDatImpl odkaz na instanci třídy SessionFactory. Je tedy vytvořen konstruktor, který obsahuje jediný atribut právě SessionFactory.

```
public class CommentDaoImpl implements CommentDao {
    private SessionFactory sessionFactory;
    public CommentDaoImpl(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
```

O úspěšné vytvoření instance třídy CommentDaoImpl se stará samotný Spring framework. V aplikačním kontextu je nastavena beana s cestou k implementující třídě a předaným parametrem s instancí sessionFactory.

```
<bean id="sessionFactory"
  class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="configLocation"
            value="classpath:hibernate.cfg.xml" />
</bean>
<bean id="commentDao"
      class="cz.soma.portal.kooperace.dao.CommentDaoImpl">
      <constructor-arg>
            <ref bean="sessionFactory" />
      </constructor-arg>
</bean>
```

Touto konfigurací můžeme pomocí anotace @Autowired zajistit, aby se framework postaral o správné nainstancování a referenci na tuto CommentDaoImpl třídu například v kontroleru. Zde pak můžeme volat metodu z CommentDaoImpl například pro uložení komentáře.

```
@Override
public boolean saveComment(Comment comment) {
	boolean retVal = false;
	Session session = sessionFactory.getCurrentSession()
		.getSessionFactory().openSession();
	try {
		session.beginTransaction();
		session.save(comment);
		session.getTransaction().commit();
		retVal = true;
	} finally {
		PortalUtils.closeSession(session);
	}
	return retVal;
}
```

Metoda saveComment vrací boolean informaci o úspěchu nebo neúspěchu uložení komentáře do databáze. Obdobným způsobem jsou řešeny všechny ostatní metody v třídách obsluhujících ostatní modelové třídy.

5. Nasazení portálu do provozu

Pro nasazení portálu pro dodavatele ve firmě SOMA je využit stávající firemní server určený pro webové aplikace. V současnosti na tomto serveru běží Apache Tomcat a na něm webový portál pro zákazníky (servisní portál). Server je ze síťového hlediska umístěn v takzvané demilitarizované zóně, tzn. že standardně nemůže přistupovat na ostatní servery, protože je umístěn v jiné virtuální LAN síti (VLAN). Server je přístupný ze sítě internet a proto je vhodný pro nasazení zde řešeného kooperačního portálu.

5.1 Nastavení přístupů do sítě – iptables

Přístup serveru webového portálu do vnitřní sítě je povolen pouze ve dvou případech. Jedná se o přístup na databázový server přes port 3050 a na server, který poskytuje Pdf výkresy a data k výkresům přes port 80. Díky první výjimce je možné portál připojit do stávající databáze a zpřístupnit tak některá data ze sítě internet. Druhá výjimka umožňuje, aby portál mohl stahovat výkresy (data) z firemního serveru a zobrazovat je tak uživatelům kooperačního portálu. Tento přístup je řešený na úrovni firemního firewallu, který je nastavován Linuxovým nástrojem iptables.

Nástroj iptables slouží v linuxových systémech pro práci se síťovou komunikací. Iptables umožňuje nakonfigurovat různé druhy firewallů (stavový, transparentní, …), nakonfigurovat vlastní NAT nebo sdílení internetu. Pravidla se v nástroji definují pomocí příkazu „iptables" přes příkazový řádek. Pokud paket, který putuje přes firemní firewall, tomuto pravidlu vyhoví, další pravidla se již netestují a s paketem je provedena požadovaná akce (ACCEPT nebo DROP). Výchozí politika v iptables je, že pokud paket nevyhovuje žádnému pravidlu, tak je komunikace povolena.

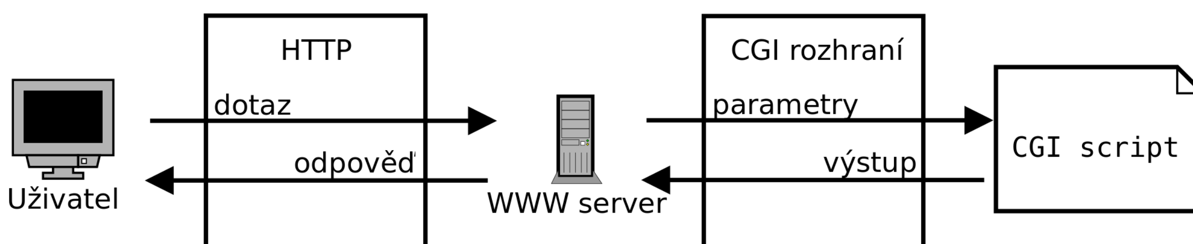
5.2 Stahování dat z vnitřní sítě – CGI skripty

V předchozích kapitolách je zmíněna možnost webového portálu stahovat výkresy a data k výkresům přímo z vnitřní firemní infrastruktury. To je, jak již bylo řečeno, řešeno pomocí povoleného přístupu přes port 80 na server, který umožňuje obsluhovat požadavky na výkresy pomocí url požadavků. Tyto požadavky jsou obsluhovány pomocí tzv. CGI skriptů.

CGI skript je vlastně spustitelný soubor na serveru, který vrací nějaký vygenerovaný kód v různých formátech (HTML, XML, Pdf, ...). Pro psaní CGI skriptu je možné použít různé programovací jazyky (různé interprety Unixu, Perl, C nebo C++). Aby byl výsledný program použitelný jako CGI skript, musí splňovat pouze dvě podmínky (musí umět přebírat parametry pomocí rozhraní CGI a výsledek činnosti skriptu musí být ve formátu HTTP zapsán na standardní výstup programu). [KOSEK]

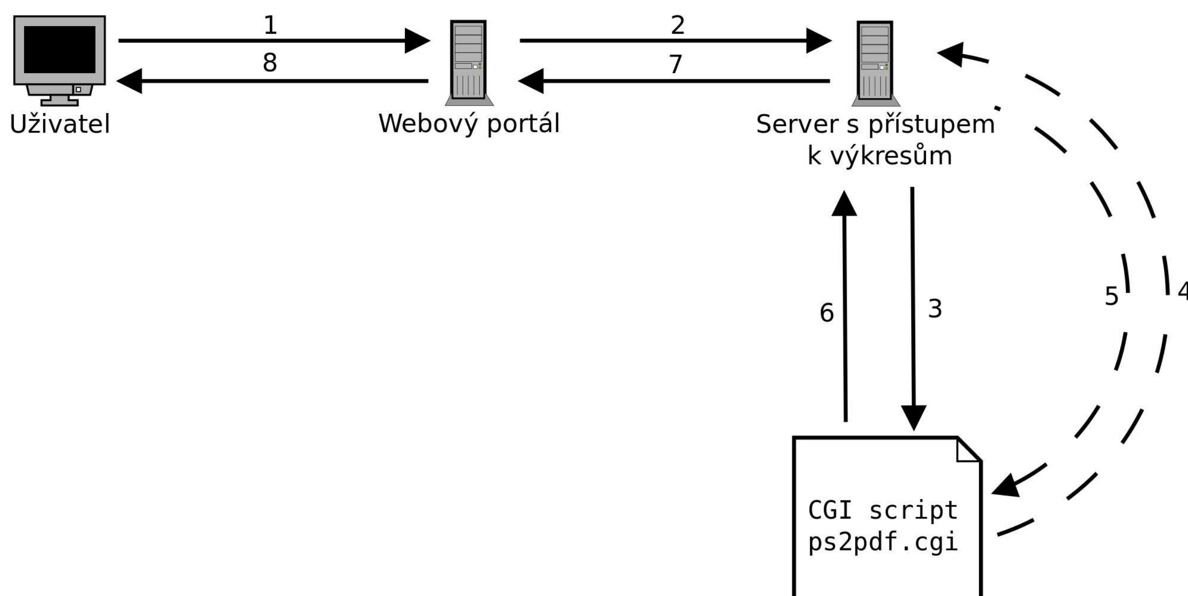
5.2.1 Spolupráce mezi WWW serverem a CGI skriptem

Na začátku komunikace je požadavek klienta na určité URL. Toto URL však nyní ukazuje na CGI skript. Kromě adresy CGI skriptu může klient předat serveru parametry, které ovlivní chování CGI skriptu. Server zjistí, že požadované URL je potřeba obsloužit CGI scriptem. Server to nejčastěji pozná podle toho, že na začátku cesty v URL je uveden adresář cgi-bin. Nyní hlavní slovo dostává CGI skript. Zpracuje předané parametry a jako výsledek na svůj standardní výstup vypíše odpověď ve tvaru protokolu HTTP. Tento výstup je zachycen WWW serverem, který doplní další hlavičky do HTTP odpovědi a vše pošle zpět klientovi jako odpověď (viz obrázek 19). [KOSEK]



Obrázek 19: Obsloužení požadavku CGI skriptem. [KOSEK]

Konkrétní situace však neodpovídá ukázkovému případu (obrázek 19). Dle firemní infrastruktury je požadavek na zobrazení Pdf verze výkresu obsluhován dle následujícího schématického obrázku 20.



Obrázek 20: Schéma stahování výkresů pomocí CGI skriptu. [autor]

Popis jednotlivých kroků k obsloužení požadavku na výkres:

1. Požadavek uživatele portálu na stažení konkrétního výkresu.

2. Požadavek webového portálu přes HTTP (port 80) obsahující název CGI skriptu a výkresu.
3. Volání scriptu ps2pdf.cgi (převod formátu postskript do Pdf) s parametrem název výkresu.
4. Čtení souboru s výkresem a převod do Pdf formátu.
5. Čtení souboru s výkresem a převod do Pdf formátu.
6. Výpis soubor na standardní výstup skriptu ps2pdf.
7. Zaslání Pdf dat zpět do webového portálu.
8. Uložení vrácených dat do souboru a odeslání uživateli.

Stejným způsobem se stahují data k výkresům, jen pomocí jiných skriptů, které data nepřevádějí na Pdf ale pouze je předají tak jak jsou.

6. Shrnutí výsledků

Stanovený cíl práce o návrhu a implementaci webového portálu byl splněn. Webový portál rozšiřuje dosavadní aplikační portfolio firmy SOMA a umožňuje tak přístup k datům také externím dodavatelům. Návrh probíhal v souladu s oddělením nákupu a kooperace, kdy již v průběhu vývoje měli vedoucí pracovníci těchto oddělení možnost zasahovat do průběhu a upřesňovat svoje představy a požadavky na funkčnost. Implementovány byly nakonec všechny potřebné funkce, které byly požadovány.

Po dokončení první fáze vývoje se webový portál skutečně nasadil do ostrého provozu. Portál firmě SOMA přinesl možnost, jak některá svoje data zpřístupnit externím dodavatelům. Tuto možnost ocení zejména menší dodavatelé, kteří nemají vlastní informační systém. U větších dodavatelů totiž dochází k tomu, že objednávky jsou realizovány v rámci automatické komunikace mezi informačními systémy pomocí Csv souborů dle předem definovaného formátu. U menších dodavatelů bez této možnosti automatické komunikace usnadní webový portál komunikaci ohledně termínů jednotlivých objednávek, stahování dokumentace a dalších často řešených situací.

Před nasazením webového portálu docházelo často k situaci, že data přikládaná do mailové objednávky překračovala velikost několika MB a někdy se stávalo, že mail byl nedoručitelný. Cestou od poštovního serveru SOMA k poštovnímu serveru dodavatele mohla zpráva narazit na omezení velikosti a tak byla vrácena jako nedoručitelná. Tato situace nastávala zejména u objednávek, které obsahovaly velké množství laserových operací, vyžadujících CNC data a modely, které se přikládali k objednávce. Nyní tuto situaci řeší webový portál, který je schopen z firemního datového úložiště do webového prohlížeče dodavatele stahovat tato data v komprimované podobě, pohodlně a rychle (desítky souborů v řádu několika málo vteřin).

Usnadnění také nastává u materiálových objednávek, kde současná praxe funguje tak, že po odeslání objednávky musí dodavatel objednávku potvrdit (zpět zaslal termínové potvrzení mailem nebo telefonoval přímo s pracovníkem nákupního oddělení). Nyní může k tomuto potvrzení pohodlně využít webového portálu. V situaci, že souhlasí se všemi navrhovanými termíny dodání stačí pouze vyhledat požadované číslo objednávky a jedním stiskem tlačítka všechny termíny potvrdit.

7. Závěr a doporučení

Diskuse a návrh rozšířit aplikační infrastrukturu firmy SOMA o kooperační portál probíhala již od léta roku 2015. V té době se ve firmě odkláněla výroba dílů na externí firmy ve větší míře, než je tomu o téměř rok později na jaře 2016. Objem kapacitních odklonů během neceléhoroku klesl téměř na nulu a celkový objem kooperace klesl zhruba na jednu třetinu. Proto i vývoj webového portálu musel nepatrně změnit svoje původní zaměření. Během vývoje se rozhodlo, že se nebude jednat pouze o portál pro kooperační firmy, ale také pro dodavatele materiálu. Tato změna způsobila, že dnes je webový portál určen pro širší škálu dodavatelů a není omezen pouze pro užší počet dodavatelských firem. Pokles počtu odklonů výroby dílů na externí firmy se dle vyjádření vedení firmy nepředpokládá trvale, ale jedná se pouze o dočasnou situaci. Díky této situaci větší míra využívání webového portálu nastane až tehdy, kdy bude portál nasazen delší dobu, bude více otestován a odladěn od chyb, které při vývoji vznikly, a které se objevují až při ostrém využívání skutečnými uživateli portálu.

Odladěním chyb a nedostatků vývoj portálu nekončí. Po delším čase testování ze strany dodavatelů a odstraněním nedostatků se může portál dále rozšiřovat. Určitě se už dnes nabízí mnoho dalších funkcí a rozšíření, které by mohl portál obsahovat. Určitě by stálo za zvážení optimalizovat portál více na mobilní zařízení. Mohly by přibýt funkce, kde by si uživatel nastavoval požadovaný formát objednávky a jaké informace by měla objednávka obsahovat. Mnoho větších dodavatelů by určitě uvítalo možnost zadat k objednávaným materiálům svoje ID čísla, aby mohl požadavky firmy SOMA lépe a rychleji vyhledat, tato možnost dnes existuje pouze v interním informačním systému, kam mohou přistupovat pouze zaměstnanci firmy SOMA.

8. Seznam použité literatury

1. [CUBRID] Understanding Java Garbage Collection, Sangmin Lee, 2012. (citace březen 2016). Přístup z internetu:
   URL: http://www.cubrid.org/blog/dev-platform/understanding-java-garbage-collection/

2. [FIREBIRD] About Firebird. Firebird Foundation Incorporated, 2016. (citace březen 2016). Přístup z internetu:
   URL: http://www.firebirdsql.org/en/about-firebird/

3. [IB-AID] Firebird 2.5 Language Reference, Beta Release 1, Leden 2016. Dmitry Filippov, Alexander Karpeykin, Alexey Kovyazin, Dmitry Kuzmenko, Denis Simonov, Paul Vinkenoog, Thomas Woinke, and Dmitry Yemanov. Verze 0.900. (citace únor 2016). Přístup z internetu:
   URL: http://ib-aid.com/download/docs/firebird-language-reference-2.5-english.pdf

4. [IB-EXPERT] Multi-generational architecture (MGA) and record versioning, IBExpert, 2014. (citace únor 2016). Přístup z internetu:
   URL: http://www.ibexpert.net/ibe/index.php?n=Doc.Multi-generationalArchitectureMGAAndRecordVersioning

5. [ITNETWORK] MVC architektura (online). Itnetwork.cz, David Čápka, 2016. (citace únor 2016). Přístup z internetu:
   URL: http://www.itnetwork.cz/navrhove-vzory/mvc-architektura-navrhovy-vzor/

6. [JAVAHONK] How many types memory areas allocated by JVM (online). Java Honk, 2014. (cit. 20.12.2015). Přístup z internetu:
   URL: http://javahonk.com/how-many-types-memory-areas-allocated-by-jvm/

7. [KOSEK] Aplikace na Webu: 6. CGI-skripty, Jiří Kosek ml., 1999. (cit. 20.3.2016). Přístup z internetu:
   URL: http://www.kosek.cz/clanky/iweb/06.html

8. [KOTACKA] Gradle, moderní nástroj pro automatizaci, Vít Kotačka, zdroják.cz, 7.6.2013. (citace březen 2016). Přístup z internetu:

URL: https://www.zdrojak.cz/clanky/gradle-moderni-nastroj-na-automatizaci/

9. [KHORSUN] KHORSUN, Vlad. FIREBIRD DEVELOPERS TEAM. *Garbage collection in Firebird*. Firebird Tour 2013, Praha, 2013. Prezentace na konferenci Firebird tour 2013.

10. [KUZMENKO] KUZMENKO, Dmitry. IBSURGEON. *Transactions in Firebird: Multi-versioning, Transaction parameters, Transaction inventory*. Firebird Tour 2013, Praha, 2013. Prezentace na konferenci Firebird tour 2013.

11. [MAVEN] Apache Maven Project, What is Maven. Březen 2016. (cit. Březen 2016). Přístup z internetu:
URL: https://maven.apache.org/what-is-maven.html

12. [MCNAB] MCNAB, Chris. Network security assessment. 2.vyd. Sebastopol: O'Reilly Media, Inc., 2008. ISBN 0596510306.

13. [ORACLE] Java 8 information from the Oracle Technology Network (online), Oracle, 2014. (cit. 15.1.2016). Přístup z internetu:
URL: http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html

14. [OTTINGER] OTTINGER, Joseph B, Dave MINTER a Jeff LINWOOD. Beginning Hibernate. 3.vyd. New York: Apress, 2014. ISBN 9781430265177.

15. [OWASP] Cross-Site Request Forgery (CSRF), owasp.org, 14.10.2015. (citace březen 2016). Přístup z internetu:
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

16. [SUN] Building Web Components, Design Pattern and Frameworks, Sun Microsystems, Inc, 2002. Přístup z internetu:
URL: https://docs.oracle.com/cd/E19929-01/816-4337/03_design_issues.html

17. [TOMCAT] Apache Tomcat Versions, Marc A. Saegesser, Yoav Shapira, Jean-Frederic Clere, 2016. (citace březen 2016). Přístup z internetu:

URL: http://tomcat.apache.org/whichversion.html

18. [VARANASI] VARANASI, Balaji a Sudha BELIDA. Introducing Maven. Berkeley, CA: Apress, 2014. Expert's voice in Java. ISBN 1484208420.

19. [VUKOTIC] VUKOTIC, Aleksa a James GOODWILL. Apache Tomcat 7. 1. vyd. New York: Apress, 2011. ISBN 978-143-0237-235.

20. [WALLS] WALLS, Craig. Spring in action. Fourth Edition. Shelter Island, NY: Manning, 2015. ISBN 161729120X.

21. [WINCH] WINCH, Robert a Peter MULARIEN. Spring Security 3.1: secure your web applications from hackers with the step-by-step guide. Birmingham, UK: Packt Pub., 2012.

22. [ZIZKA] Začátky se Spring AOP, Ondřej Žižka. (citace prosinec 2015). Přístup z internetu:
URL: http://ondra.zizka.cz/stranky/programovani/java/navod-spring-aop-zacatky.texy

9. Přílohy

1. CD se zdrojovými kódy a náhledy webové aplikace

Obsah přiloženého CD:

/screenshots/ - adresář obsahuje náhledy důležitých obrazovek webové aplikace,

/source-code/ - adresář obsahuje zdrojové kódy aplikace, včetně SQL scriptu na vytvoření pohledů v databázi a konfiguračních souborů jednotlivých frameworků.

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Beran Tomáš	U Velorexu 1344, Žamberk	I14278

TÉMA ČESKY:

Webový portál pro dodavatele firmy SOMA spol. s r. o.

TÉMA ANGLICKY:

Web portal for suppliers of SOMA company

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Ve firmě SOMA spol. s r. o. zavést funkční webový portál pro komunikaci s dodavateli, který bude rozšiřovat stávající interní informační systém.

Osnova:
1. Úvod
2. Současný stav zpracování objednávek
3. Návrh webového portálu s popisem použitých technologií
4. Popis použitých metod a technického řešení portálu
5. Nasazení do běžného provozu a sledování stavu používání
6. Zhodnocení celkového přínosu a porovnání s původním stavem
7. Závěr
8. Literatura

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: .... **Datum:** 8. 10. 2015

Podpis vedoucího práce: .... **Datum:** 8.10.2015