Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Realizace on-line virtuálních skladů pro podniky v rámci EU

Plzeň, 2014 Ondřej Trhoň

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. 5. 2014

Ondřej Trhoň

Abstrakt

Tato práce se zabývá definováním specifikace požadavků na webový systém pro řízení zásob klientů v různých zemích Evropské unie, jeho návrhem a realizací. Výsledný produkt má plnit úlohu virtuálního skladu. V rámci práce dojde k seznámení s tímto termínem, popisem nejpoužívanějších technologií při tvorbě webových aplikací, specifikování a navržení požadovaného systému s ohledem na zvolenou technologii (ASP.NET MVC), vlastní implementaci a testovací nasazení.

Abstract

The purpose of this thesis is to define software requirements specification of web application to controlling the movement and storage of customer resources anywhere in the European Union, and to define its design and implementation. The final product will work as a virtual inventory. This thesis contains an introduction to this term, a description of the most popular technologies used to develop web applications, a specification and a draft of the desired product using the chosen technology (ASP.NET MVC), the implementation itself and a test deployment.

Poděkování

Chtěl bych poděkovat své rodině za podporu a pomoc při studiu. Svým kolegům a bratrovi za cenné technické rady a připomínky. A své přítelkyni za oporu.

Obsah

1 Úvod

V dnešní době je při prodeji zboží kladen velký důraz na cenu zboží a cenu vedlejších nákladů jako dopravné, skladné, balné a bankovní poplatky. Jedním z nejdůležitějších faktorů při prodeji zboží je dodací termín. Zákazníci již nejsou ochotni čekat na zboží několik týdnů a očekávají dodávky „just-in-time“. V rámci EU odpadla díky celní unii povinnost proclít zboží, pročež se placení celních poplatků a daní v zemi kupujícího stává minulostí. Otevřely se tak možnosti vstupu na zahraniční trhy i pro malé a střední firmy. Nastává zde ovšem problém s cenou dopravy, která je nesrovnatelná s náklady na dopravu v tuzemsku. Nejsou zde dopravní společnosti, které umí přepravit zboží na dobírku do druhé země. Např. Česká pošta nemá smlouvu se Slovenskou poštou o výběru hotovosti. Pokud se nějaká taková firma najde, je toto velmi drahé.

V případě většího prodeje např. na výše zmíněné Slovensko se nabízí možnost zřízení tamní pobočky a vedení samostatného skladu. To s sebou ale přináší další starosti – firmu může v cizí zemi založit fyzická osoba, ale musí zde mít trvalý pobyt. Z tohoto důvodu je nutné založit tzv. právnickou osobu, složit základní kapitál a poté zajistit zaměstnance a prostory pro podnikání. Vše uvedené s sebou nese zvýšené náklady v podobě nájemného, platby sociálního a zdravotního pojištění a nutnost vedení účetní evidence dle zákonů příslušného státu.

Pravděpodobně lepší možností je využití tzv. „virtuálních skladů“, o kterých se zmiňuje anglický podnikatel Richard Koch ve své knize The 80/20 Principle [1]. Nejedná se o pouhý pronájem skladového prostoru, ale o jeho „virtualizaci“, kdy svůj sklad člověk řídí on-line skrze webové rozhraní. Tento systém má minimalizovat náklady na obou stranách – jak na straně firmy, která virtuální sklady využívá, tak poskytovatele těchto skladů, který se může soustředit na předmět svojí činnosti – outsourcing skladování, balení a expedování – a dělat to podstatně levněji.

Cílem této práce je vytvořit takový systém, který bude umožňovat správu virtuálních skladů pro jednotlivé zákazníky v Čechách, na Slovensku a v Polsku a bude snadno rozšířitelný o další funkce (např. usnadnění práce pomocí čtečky čárových kódů, zobrazení kamerových záznamů z balení, ...), bude integrován do stávající firemní infrastruktury a v případě, že nebude na nějaký systém přímo napojen, bude jej nahrazovat (např. systém pro správu novinek, akcí apod.). Výsledná webová aplikace by měla umožňovat přepínání do daných jazyků a filtraci dat. Velký důraz by měl být kladen i na uživatelské ovládání, které musí být jednoduché a intuitivní.

2 E-logistika

Nejprve je nutné definovat logistické řízení jako takové. V [2] je logistické řízení definováno jako proces plánování, realizace a řízení efektivního, výkonného toku a skladování zboží, služeb a souvisejících informací z místa vzniku do místa spotřeby, jehož cílem je uspokojit požadavky zákazníků.

V dnešní době se díky internetu a jeho potenciálu rozšířil trh i do elektronické podoby, což dalo vzniknout novému odvětví v podnikání – e-businessu.

E-logistika je logistický systém, který je integrální součástí e-businessu. V jeho rámci zajišťuje logistické toky pro potřeby dalších dílčích částí e-businessu jako například e-commerci. Charakteristickým rysem e-logistiky je sběr, přenos, zpracování a uchování dat a informací za využití informačních technologií jako jsou například internet nebo intranet. [3]

Všechny procesy v distribuční firmě můžeme rozdělit na procesy, které provádí vlastní logistickou činnost jako je přeprava zboží, skladování zboží, atd., a procesy, které přinášejí informace jako administrativu, marketing a prodej služeb. Tyto procesy lze převést do elektronické podoby a zpracovávat za pomoci e-business aplikací. Tím může dojít k podstatnému snížení celkových nákladů. Profesor Frankel z Massachusetts Institute of Technology (MIT) uvádí hodnotu až 40%. Z výzkumu mezinárodního obchodu provedeného na MIT vyplynulo, že z celkové hodnoty logistických služeb ve výši 1 billionu USD tvoří hodnotu služeb, které lze převést do e-podoby, přibližně jednu třetinu. [4]

2.1 Supply Chain Management (SCM)

SCM, neboli řízení dodavatelského řetězce, je označením pro systémy, prostředky a postupy, které slouží pro koordinaci materiálů, výrobků, služeb, informací a financí, které plynou od dodavatelů surovin přes zpracovatele, výrobce, velkoobchodníky a maloobchodníky, až ke spotřebitelům. Celý proces začíná zadáním objednávek, jejich posouzením a zpracováním, pokračuje výrobou a dodáním zboží a služeb a končí zpětnou vazbou. Cílem SCM je dosažení efektivního využití všech zdrojů vstupujících do procesu, včasné dodání všech výrobků a služeb, rychlost procesu, minimalizace prostojů a nulové ztráty. [5]

Dle definice SCOR (Supply Chain Operation Model) je možné SCM rozdělit do pěti základních částí [6]:

1. **Plánování** – strategická část SCM nutná k řízení všech zdrojů směrem k naplnění požadavků zákazníka na výrobek nebo službu. Součástí je definice sady metrik k monitorování celého řetězce.

2. **Nákup** – výběr dodavatele, materiálu, resp. služeb potřebných pro realizaci vlastní produkce. Součástí je ocenění dodávky, dodací a platební podmínky a následné monitorování tohoto vztahu.

3. **Výroba** – výroba, rozvrhování činností a operací nutných pro výrobu, testování, balení a příprava expedice. Tato část je nejvíce náročná na měření kvality a produktivity zaměstnanců.

4. **Distribuce** – část řetězce, která je mnohými označována jako logistika. Koordinuje příjem zakázek od zákazníka, využívá sklady a transportní možnosti k dodání produktu zákazníkovi. Stará se o fakturaci a placení.

5. **Reklamace** – část řetězce, která zajišťuje příjem nesprávného zboží od zákazníka a řeší potíže zákazníků s dodávkami.

Pohled na lineární vazby ve velmi jednoduchém dodavatelském řetězci:

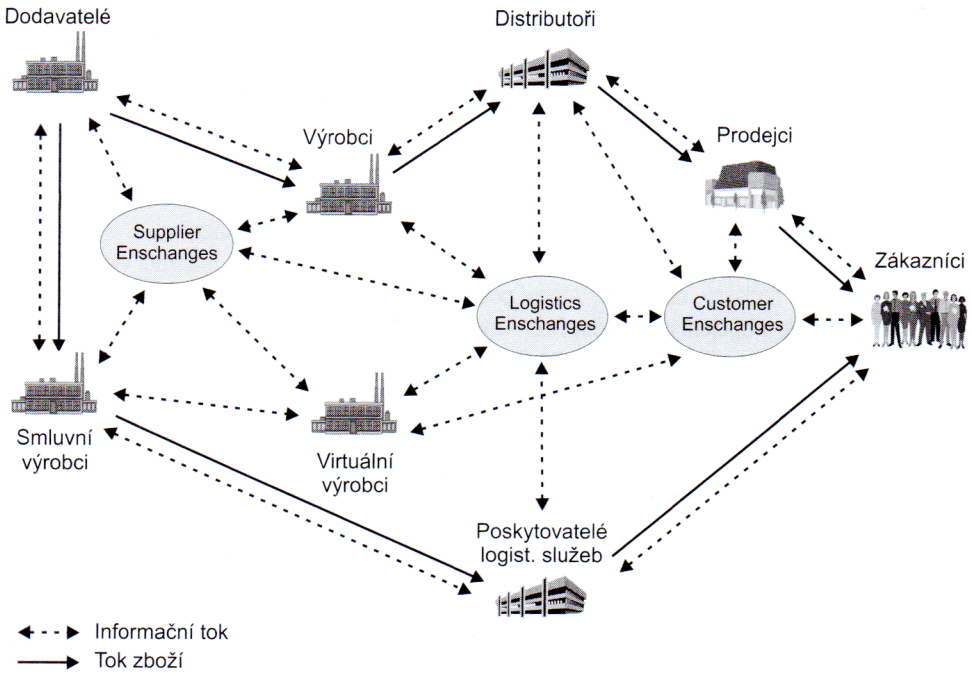
dodavatel → výrobce → distributor → prodejce → zákazník

Tok zboží směřuje od dodavatele ke konečnému zákazníkovi. Přesně naopak směřuje tok informací a finančních prostředků.

Ve skutečnosti ale mohou být vazby v dodavatelském řetězci mnohem složitější, zvláště po nástupu internetu, viz obrázek 1.

2.2 Warehouse Management System (WMS)

Pojem WMS označuje systém pro řízení skladů a bývá klíčovou součástí SCM. Tento systém umožňuje centralizovanou správu skladů, včetně řízení zásob, sledování a umístění zboží, a také podporu každodenního provozu skladu. Systém WMS není jen uzavřenou soustavou, ale umožňuje komunikovat s okolním světem. WMS se může použít jako samostatná aplikace nebo být nedílnou součástí většího systému jako je ERP (Enterprise Resource Planning) systém. Hlavním důvodem integrování WMS přímo do ERP je především snaha o poskytnutí kompletního řešení. Mezi ERP systémy, které řeší skladování zásob, patří např. SAP, Microsoft Dynamics AX a Epicor Software.



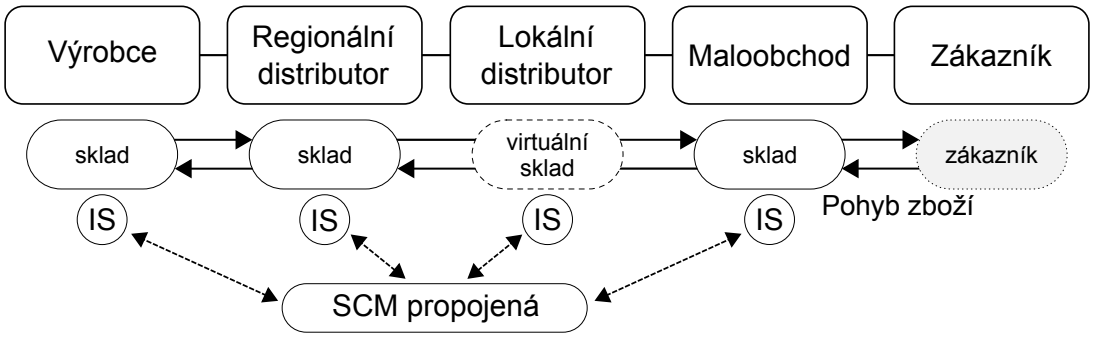
Obrázek 1: Příklad schéma dodavatelského řetězce na bázi internetu. [6]

V posledních letech se velmi rozšířily společnosti poskytující logistické služby ve formě outsourcingu. Poskytovatel těchto služeb většinou požaduje, aby skladovací systém byl flexibilní a umožňoval rychle reagovat na požadovanou změnu ve skladu, a to za velmi krátkou dobu. Běžně se nabízí napojení na nějaký typ TMS (Transportation Management System), na různé finanční aplikace a další systémy klientů, kterými můžou být i e-shopy. V praxi má většinou každý klient svůj specifický informační systém, který odesílá a přijímá data v různých formátech. [7]

Pro toto specifické využití WMS se občas používá označení virtuální sklad.

2.2.1 Virtuální sklady

Skutečné sklady lze v dodavatelském řetězci nahradit sklady virtuálními, viz obrázek 2. Firmy mohou prostřednictvím řetězce dodávat zboží přímo od výrobce rovnou k zákazníkovi bez zbytečného přesouvání mezi různými sklady. Ostatní účastníci vstupují do řetězce pouze virtuálně. Pomocí informačních systémů si mezi sebou předávají jen informace o produktech. Zboží zůstává fyzicky uložené v jednom skladu a nedochází tak k zvyšování nákladů za dopravu mezi dalšími sklady. [3]



Obrázek 2: Schéma SCM s využitím virtuálního skladu. [3]

Na virtuální sklady se dá pohlížet i jinak. Mohu sloužit ke správě skladových zásob umístěných ve fyzických skladech, které se nachází na různých místech. Takový systém pak umožňuje majiteli kontrolovat veškeré jeho zboží prostřednictvím jediného systému. Virtuální sklad vlastně vytváří abstraktní prostor pro jednodušší správu zásob ve všech užitých skladech.

Jak již bylo zmíněno v úvodu, ušetří se tak především náklady na dopravu, jelikož se podle zvolených požadavků využije takových skladů, které jsou k zákazníkovi blíže. Typicky se jedná o sklady umístěné v zahraničí pro zajištění dostupnosti zboží v dané zemi.

Nástroje pro správu virtuálních skladů bývají klientům nejčastěji poskytovány skrze webové rozhraní a často bývají propojené s e-shopy. Příkladem již existujícího nástroje je např. slovenská služba PackService[1].

[1] `http://www.packservice.sk`

3 Vývoj webových aplikací

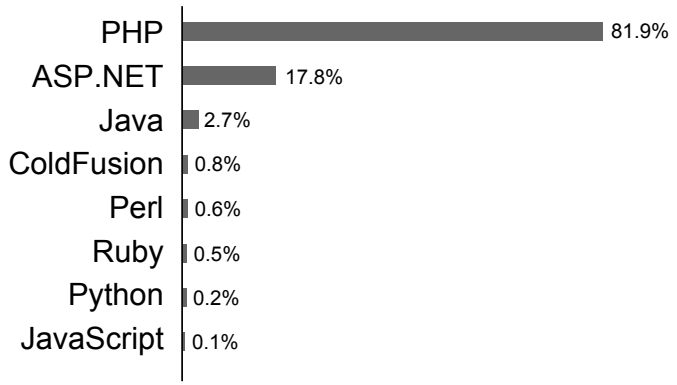
Tato kapitola popisuje základní prostředky, které se využívají při vytváření webových aplikací a které jsou potřeba pro jejich běh. Za webovou aplikaci se považuje aplikace, která je poskytována uživatelům prostřednictvím internetu nebo intranetu a běží na webovém serveru. Uživatelé ji mohou používat prostřednictvím webového prohlížeče, který plní úlohu tzv. tenkého klienta.

Webové stránky, které tenký klient zobrazuje, jsou tvořeny zdrojovým kódem – ten je typicky psaný pomocí značkovacího jazyka HTML/XHTML rozšířeného případně o kód ve skriptovacím jazyce, java applety, RIA (Rich Internet Application), ActiveX apod.

3.1 Programovací jazyky

Pro tvorbu dynamických webových systémů se využívají programovací jazyky, které jsou prováděny na straně serveru (server-side) a uživateli je posílán až výsledek. Tudíž si je uživatel nemůže zobrazit a teoreticky ani nemusí mít možnost zjistit si, pomocí jakých technologií je daný web tvořen.

Na obrázku 3 je vidět procentuální zastoupení server-side programovacích jazyků dle W3Techs k 7. dubnu 2014.



Obrázek 3: Procento zastoupení server-side programovacích jazyků. [8]

3.1.1 PHP

Jedná se o velmi rozšířený skriptovací programovací jazyk, který vznikl v roce 1995 a momentálně se nachází ve verzi 5.5. Jeho součástí jsou knihovny pro snadné připojení k většině nejrozšířenějších databázových systémů (typicky MySQL). Pro svůj běh vyžaduje webový server (typicky Apache).

Při vývoji rozsáhlejších aplikací se často využívá v kombinaci s některým z frameworků (Zend, Nette, ...), které jsou buď zdarma nebo i placené, a které takový vývoj zpřehledňují a urychlují.

Zástupci: i velmi rozsáhlé projekty jako `Yahoo.com`, `Facebook.com` či `Wikipedia.org`.

Výhody:

- Multiplatformnost,
- rozšířenost a podpora na hostingových službách,
- mnoho již existujících produktů, které jsou distribuovány zdarma i pro komerční užití,
- existence frameworků, které řeší některé nevýhody (snížení bezpečnostních rizik, debugování),
- jednoduchost.

Nevýhody:

- Slabá typová kontrola,
- nekonzistentní názvy funkcí,
- v základu chybí debugovací nástroj,
- problematická zpětná kompatibilita při vydání nových verzí,
- primitivní zpracování chyb (ve srovnání s ostatními jazyky).

3.1.2 Java

Java bývá nejčastěji spojována s vývojem velkých webových aplikací. V praxi je ale vhodná i na menší a střední weby, které se mohou později rozšiřovat. Existuje pro ni mnoho frameworků, které za programátora řeší časté základní konstrukce.

Pro vývoj a provoz podnikových aplikací a informačních systémů slouží Java Platform, Enterprise Edition (Java EE). Její součástí jsou servlety, které zpracovávají dotazy a vykonávají vnitřní logiku, a JavaServer Pages (JSP), které reprezentují view vrstvu.

Jednou z nejpoužívanějších architektur pro vývoj webových aplikací pomocí Java EE je Model-View-Controller (MVC). Přičemž servlet slouží jako controller,

JSP jako view a běžná třída jako model. Model může být implementovaný jako tzv. JavaBean a usnadnit tak přenos dat mezi jednotlivými vrstvami (což vyžaduje dodržovat zavedené konvence pro JavaBean).

Pro svůj běh vyžaduje tzv. servletový kontejner, neboli aplikační server (např. Apache Tomcat). Ukázka putování požadavku od klienta do servletu, vč. následné odpovědi v podobě HTML stránky, je naznačen na obrázku 4.

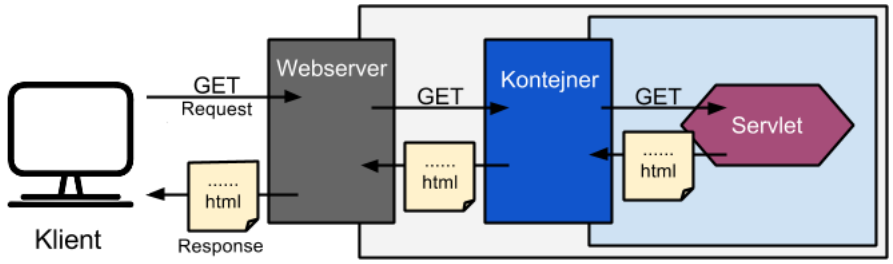
Zástupci: např. `eBay.com`, `Amazon.com` nebo sociální síť `LinkedIn.com`.

Výhody:

- Menší náchylnost na chyby díky statické typové kontrole (na rozdíl od PHP),
- rozdělení do více vrstev usnadňuje testování,
- velké množství různých rozšíření a frameworků,
- škálovatelnost.

Nevýhody:

- Nepříliš rozšířený hosting,
- projekt potřebuje upravit pro použití v jiném servletovém kontejneru.



Obrázek 4: Ukázka zpracování GET požadavku pomocí servletu.

3.1.3 ASP.NET

Nástupce scriptovací platformy Active Server Pages (ASP) pro vytváření dynamických webových stránek založený na .NET Frameworku. Stejně jako Java bývá tato technologie spojována s vývojem velkých webových aplikací.

ASP.NET využívají tzv. WebForms, kde je jako základ brán webový formulář (stránka). Každá stránka může mít code-behind, který slouží pro oddělení kódu

a designu stránek. Při vývoji se dají používat webové kontrolky (základní nebo i vlastní), velmi podobné jako pro desktopové aplikace.

Jelikož je HTTP protokol bezstavový, tak si formulář i všechny kontrolky drží informaci o svém stavu pomocí ViewState. Informace o stavu se posílá klientovi v HTML stránce zakódovaná ve formátu Base64 – díky čemuž dojde ve většině případech k značnému nárůstu velikosti takové stránky, pročež je ViewState považována za jednu z nejkontroverznějších funkcí ASP.NET [9]. Výhodou tohoto řešení je ale redukce zátěže databáze i serveru.

Výhody:

- Možnost použít libovolný jazyk, který má podporu .NET,
- menší náchylnost na chyby díky statické typové kontrole (na rozdíl od PHP),
- rozdělení do více vrstev usnadňuje testování,
- velké množství různých rozšíření a frameworků,
- škálovatelnost.

Nevýhody:

- Cena při externím hostování,
- kontroverzní ViewState.

Zástupci: např. `Bing.com` nebo např. `Nbcnews.com`.

3.1.4 ASP.NET MVC

Webový aplikační framework založený na ASP.NET, který je zároveň jeho alternativou. Poskytuje jiný přístup k tvorbě webových projektů – na rozdíl od ASP.NET, který se snaží přiblížit programování webu desktopovým vývojářům, je ASP.NET MVC navržený přímo pro webové vývojáře. [10]

Výhody:

- Úplná kontrola nad vygenerovaným HTML (narozdíl od ASP.NET),
- ,,hezká url“ v základu (SEO a REST optimalizace),
- velká nabídka alternativních view enginů (NVelocity, Brail, NHaml, ...),

- navrženo od základu pro web a webové vývojáře (narozdíl od ASP.NET).

Nevýhody:

- Velmi problematická integrace do již existujícího ASP.NET projektu.

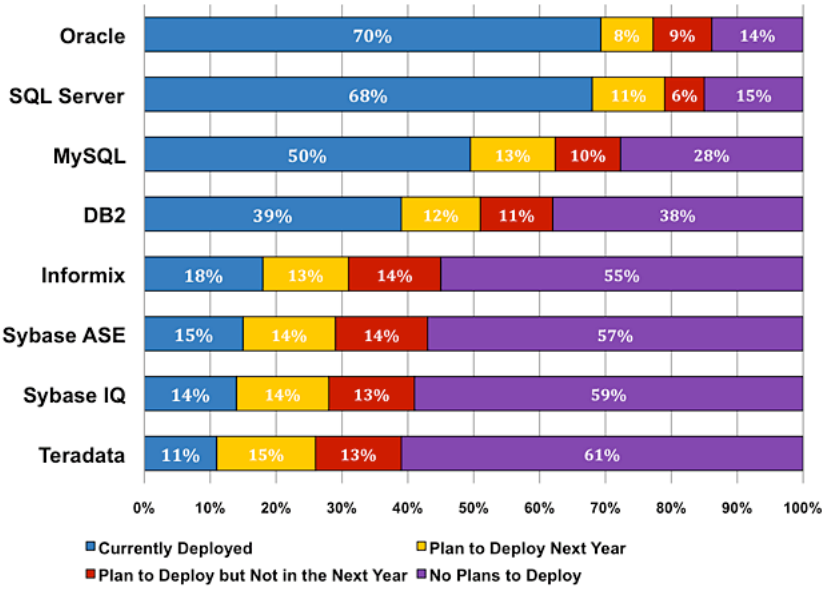
Zástupci: např. `Stackoverflow.com`, `CodePlex.com` či `Dell.com`.

3.1.5 Ostatní

Pro tvorbu webových systémů existuje celá řada dalších aplikačních frameworků, které se ve většině případů více či méně drží architektury MVC, jako je Django, Ruby on Rails, Catalyst a další.

3.2 Databáze

Tvorba dynamických webových stránek se neobejde, až na drobné výjimky, bez využití některé z databází – data je potřeba někam ukládat. Databází je myšlen systém řízení báze dat (Database Management System – DBMS) i samotná báze dat.



Obrázek 5: Zastoupení DBMS v roce 2008 dle společnosti Gartner, Inc. [11]

Existuje spousta typů databází, přičemž nejpoužívanější a nejznámější jsou relační databáze (Relational database management system – RDBMS). Na obráz-

ku 5 jsou uvedeny ty nepoužívanější v roce 2008 podle společnosti Gartner, Inc. Většina RDBMS umožňuje stejnou základní funkcionalitu.

3.2.1 Oracle Database

Lídr na poli RDBMS – díky vysokému výkonu, pokročilým možnostem zpracování dat a snadné škálovatelnosti. Je k dispozici ve třech základních verzích, přičemž verze Express je k dispozici zdarma. Omezení spočívá v restrikci použitých prostředků na 11 GB místa pro uživatelská data, 1 CPU a 1 GB RAM. Shrnutí je vidět v tabulce 1. Pro správu databáze slouží nástroj Oracle SQL Developer.

Vlastník	Oracle Corporation
Edice	Express, Standard, Enterprise
Operační systém	Multiplatformní
Zprostředkování přístupu	ODP.NET, OCI, JDBC, ODBC
Dotazovací jazyk	PL/SQL

Tabulka 1: Základní shrnutí poznatků o Oracle Database.

3.2.2 Microsoft SQL Server

V posledních letech rychle se rozvíjející RDBMS, který se snaží prosadit na všech úrovních, od malých a středních podniků až po největší korporace. Nabízí se v pěti edicích určených pro produkci. Express verze je poskytována zdarma a obsahuje obdobná omezení jako Oracle Database Express. Shrnutí viz tabulka 2. Pro spravování se využívá SQL Server Management Studio.

Vlastník	Microsoft Corporation
Edice	Express, Web, Business Intelligence, Standard, Enterprise
Operační systém	Windows
Zprostředkování přístupu	OLE DB, TDS, ADO.NET, JDBC, ODBC
Dotazovací jazyk	T-SQL

Tabulka 2: Základní shrnutí poznatků o Microsoft SQL Server 2014.

3.2.3 MySQL

Velmi populární RDBMS, který nyní vlastní společnost Oracle Corporation a vydává ho pod dvěma licencemi – bezplatnou a komerční, ve které jsou nabízena různá rozšíření a více nástrojů. Vzhledem k častému užívání v kombinaci s operačním systémem Linux, webovým serverem Apache a jazykem PHP se ujala zkratka LAMP (Linux, Apache, MySQL, PHP). V tabulce 3 jsou shrnuty základní parametry MySQL. Pro správu MySQL existuje nespočet různých nástrojů, mezi nejpopulárnější správce patří webový phpMyAdmin nebo desktopový MySQL Workbench.

Vlastník	Oracle Corporation
Edice	GPL, komerční
Operační systém	Multiplatformní
Zprostředkování přístupu	ADO.NET, JDBC, ODBC
Dotazovací jazyk	SQL

Tabulka 3: Základní shrnutí poznatků o MySQL.

3.2.4 Ostatní

V praxi je dále možné setkat se s dalšími RDBMS, jako jsou např. Firebird či PostgreSQL.

Kromě relačních DBMS existují i další řešení, jako jsou NoSQL databáze. Ty se dělí podle způsobu, jakým přistupují k ukládání dat. Mezi nejpoužívanější patří dokumentové databáze, jejichž zástupcem je v současné době velmi populární MongoDb. Dokumentové databáze slouží hlavně pro ukládání semistrukturovaných dokumentů – umožňují měnit strukturu dat za běhu. Jejich hlavní výhodou je horizontální škálovatelnost. Mezi další NoSQL databáze, které se hojně využívají, patří databáze typu key-value, které disponují primitivní strukturou pro ukládání. Mezi nejpoužívanější key-value databáze se řadí systém Redis.

Příkladem jejich užití jsou převážně sociální sítě, kde se ukládá velké množství jednoduchých dat. Výhodou je vysoká rychlost při přístupu ke konkrétním záznamům v obrovskému objemu dat. [12]

4 Analýza požadavků

4.1 Možnosti a omezení u zaměstnavatele

Technické záležitosti byly řešeny u zaměstnavatele, který tuto zakázku zprostředkoval a zaštiťoval.

Zaměstnavatelem je společnost EuroSoftworks s.r.o.[2], která poskytuje služby v oblasti IT pro zákazníky v Evropě a USA. Zabývá se vývojem softwarových aplikací pomocí moderních technologií zaměřených pro různé obory podnikání.

- Omezení:
  - Téměř exkluzivní využívání technologií společnosti Microsoft.
  - Přechod ve vývojovém oddělení od ASP.NET k ASP.NET MVC.

4.2 Možnosti a omezení u zákazníka

Zákazníkem je strojírenská firma Hobes s.r.o.[3], která se zabývá tvorbou zámků, autodílů a dalších nástrojů. V současné době zaměstnává přes 187 zaměstnanců a ročně má obrat přes 200 mil. Kč. V rámci firmy se využívá několik informačních systémů, jak pro ekonomické oddělení – SQL Ekonom[4], tak pro řízení firemních zdrojů – Microsoft Dynamics AX[5], aj.

Společnost disponuje třemi servery, které využívá pro vnitropodnikové účely. Všechny počítače běží na operačním systému Windows.

- Omezení:
  - Nevhodné internetové připojení pro vlastní hostování výsledného produktu.
  - Operační systém Windows Server 2012.

4.3 Sběr požadavků

Sběr požadavků proběhl formou šesti schůzek, které se konaly po týdnu. Během těchto schůzek došlo k seznámení se s danou problematikou a vydefinování základní funkcionality a vzhledu požadovaného systému.

[2] `http://www.eurosoftworks.cz`
[3] `http://www.hobes.cz`
[4] `http://www.softbit.cz`
[5] `https://www.microsoft.com/cs-cz/dynamics/erp-ax-overview.aspx`

4.3.1 Vize a rozsah projektu

Jak již bylo naznačeno v úvodu, jedním z cílů většiny firem je snaha se rozšířit a zvyšovat svojí míru působnosti jak na tuzemském, tak na zahraničním trhu. Firmy si však kvůli nákladům udržují pouze jeden centrální sklad. Pořízení dalších skladů ve vybraných státech se pro menší podniky jeví jako ekonomicky nevýhodné. Hlavním problémem tak zůstává přeprava zboží z centrálního skladu k zákazníkovi, která je při posílání do zahraničí výrazně nákladnější.

Řešením je využití outsourcingu a nástroje, který umožní zajistit komunikaci mezi výrobcem a poskytovatelem outsourcovaného skladu.

Požadavky na takový systém jsou následující:

- Prostředí přepínatelné do různých jazyků,
- možnost překládání údajů o produktu dle dané země,
- základní práce se zbožím – naskladňování a vyskladňování,
- základní reporty o pohybu zboží,
- odpovídající zabezpečení.

4.3.2 Typy uživatelů

Správce

Může být pouze jeden, jedná se o administrátora systému. Spravuje uživatele, zavádí nové sklady a jako jediný má možnost editovat všechny údaje.

Vrchní skladník

Každý sklad má jednoho vrchního skladníka, který je zodpovědný za běh daného skladu. Vrchní skladník může vytvářet účty pro běžné skladníky.

Skladník

Každý sklad má své skladníky, kteří se starají o příjem a expedici zboží.

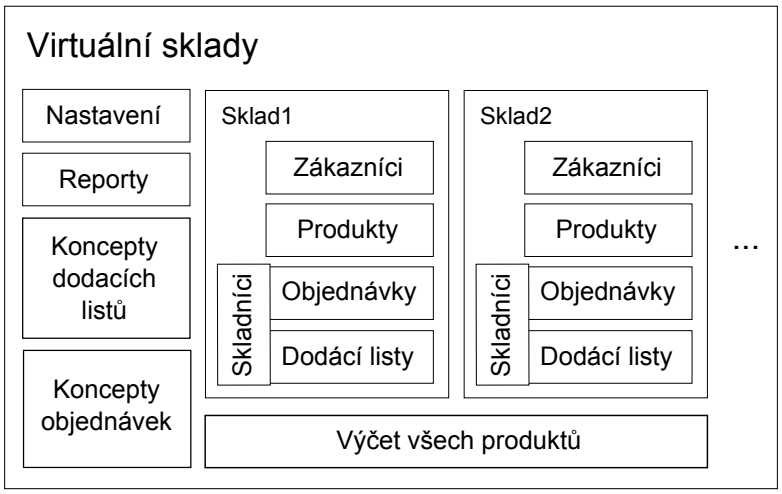
Skladník má přístup pouze k informacím, které jsou pro něj nezbytné. Nevidí ceny jednotlivých položek, pouze celé zásilky (z důvodu pojištění zásilky). Jeho primární úlohou je práce se zbožím, jeho příjem a expedice.

Vlastník zboží

Jedná se o právoplatného majitele skladovaného zboží, který pro skladování využívá některý ze skladů.

Na obrázku 6 můžete vidět základní strukturu systému z pohledu vlastníka zboží.

Vlastník zboží může vytvářet další uživatele (např. své zaměstnance), kteří budou mít stejná práva pro práci se zbožím, jako on.



Obrázek 6: Systém z pohledu konkrétního vlastníka zboží.

4.3.3 Dokument specifikace požadavků

Na základě schůzek a konzultací byl vytvořen dokument specifikace požadavků, který uvádí vše, co se od výsledné aplikace očekává. Dokument je součástí příloh umístěných na CD, viz příloha D.

4.4 Klíčové požadavky

4.4.1 Základní funkce

- **Příjem zboží na sklad** – Zajišťuje se pomocí *dodacích listů*, které vytváří přímo v systému vlastník zboží. Zboží je evidováno pro potvrzení přebrání skladníkem. Případně může skladník zapsat reklamaci.
- **Expedice zboží k zákazníkovi** – Zajišťuje se pomocí *objednávek*, které vytváří přímo v systému vlastník zboží. Je nutné specifikovat způsob dopravy a dodací podmínky (různé ceny). Skladník si objednávku rezervuje,

aby na jedné objednávce nedělalo více skladníků, zabalí a uvede do systému počet balíků připravených k expedici. Po provedení expedice a přijetí peněz označí vlastník zboží objednávku za uzavřenou.

4.4.2 Požadavky na zabezpečení

Jelikož se jedná o velmi citlivá data – při jejich kompromitování může dojít k značným finančním ztrátám, je kladen velký důraz na zabezpečení aplikace a jejich dat.

Konkrétní požadavky na zabezpečení:

- Dodržování zvolených bezpečnostních konvencí (Owasp[6]),
- logování veškerých akcí uživatelů, zvláště při pohybu zboží,
- pravidelná záloha databáze.

4.5 Možnosti nasazení

Existuje více možností pro nasazení požadovaného systému, níže je jejich porovnání podle aspektů jako je jejich variabilita, potenciál dané platformy a náklady na pořízení. Výběr je omezen na technologie společnosti Microsoft, vzhledem k požadavku na využití těchto technologií. Cena se počítá pro minimální doporučovanou sestavu určenou k produkčnímu běhu, která je uvedena ve specifikaci požadavků, včetně nákladů za případné licence. Do ceny se nezapočítávají náklady na provoz domény.

Porovnání je aktuální pro první kvartál roku 2014. Vycházelo se převážně z nabídky tuzemského trhu, zvolené příklady jsou pouze orientační.

4.5.1 Vlastní

Využití vlastního serveru pro produkční nasazení přímo u zákazníka není vhodnou volbou, hlavně kvůli omezením od poskytovatele internetového připojení. Využívané servery slouží pouze pro interní používání, případně pro občasný vzdálený přístup.

Zákazník navíc neprojevil o tuto možnost zájem.

[6] `https://www.owasp.org/index.php/Top_10_2013-Top_10`

4.5.2 Webhosting

Jedná se o pronájem místa na serveru společně s dalšími službami potřebnými pro provoz webových stránek. Zákazník nemá nad instalovaným programovým vybavením téměř žádnou kontrolu. V tabulce 4 je ukázka nabídky na českém trhu.

Společnost	**Web**	**Přibližná cena (včetně DPH)**
ZONER software, a.s.	www.czechia.com	2 400,- Kč ročně
ACTIVE24, s.r.o.	www.active24.cz	1 900,- Kč ročně
INTERNET CZ, a.s.	www.forpsi.com	2 400,- Kč ročně

Tabulka 4: Příklady cen webhostingu splňujících minimální požadavky.

4.5.3 Virtuální privátní server (VPS)

V případě VPS získává zákazník na rozdíl od webhostingu plnou kontrolu nad svým serverem a může si ho tak přizpůsobit svým potřebám. VPS běží na stejném fyzickém stroji jako virtuální server ostatních zákazníků. Výhodou tohoto řešení je snadná možnost rozšíření HW. Tabulka 5 ukazuje nabídku VPS. Do uvedených cen jsou započítány i případné další potřebné licence pro provoz požadovaných Microsoft technologií.

Společnost	**Web**	**Přibližná cena (včetně DPH)**
WEDOS Internet, a.s.	www.wedos.cz	6 800,- Kč ročně
HEXAGEEK s.r.o.	www.hexageek.cz	6 400,- Kč ročně
HOSTING90 systems s.r.o.	www.hosting90.cz	6 000,- Kč ročně

Tabulka 5: Příklady cen VPS splňujících minimální požadavky.

4.5.4 Dedikovaný server

Na rozdíl od VPS nabízí volba dedikovaného serveru pronájem fyzického HW a tím pádem garantovaný výkon. Výhodou tohoto řešení je i technická podpora, za server

zodpovídá provozovatel. V tabulce 6 jsou porovnávány ceny, do kterých jsou opět započítány i všechny potřebné licence.

Společnost	**Web**	**Přibližná cena (včetně DPH)**
WEDOS Internet, a.s.	www.wedos.cz	28 000,- Kč ročně
INTERNET CZ, a.s.	www.forpsi.com	24 000,- Kč ročně
HEXAGEEK s.r.o.	www.hexageek.cz	29 000,- Kč ročně

Tabulka 6: Příklady cen dedikovaných serverů splňujících minimální požadavky.

4.5.5 Housing

Tato služba se občas nazývá i serverhosting. V tomto případě je nutné investovat do vlastního HW, který se poté umístí do příslušného datacentra. Pronajímá se tedy pouze prostor, konektivita a energie. Případné havárie musí řešit zákazník sám. V tabulce 7 je vidět srovnání cen pronájmu, nezapočítává se cena vlastního stroje, která se u minimálních konfigurací pohybuje kolem 30 000,- Kč včetně DPH a hradí se jednorázově.

Společnost	**Web**	**Přibližná cena (včetně DPH)**
INTERNET CZ, a.s.	www.forpsi.com	19 000,- Kč ročně
Master Internet, s.r.o.	www.master.cz	20 700,- Kč ročně
WEB4U s.r.o.	www.web4u.cz	21 700,- Kč ročně

Tabulka 7: Příklady cen housingu splňujících minimální požadavky.

4.5.6 Cloud

V případě cloudu je situace trochu jiná, než u předchozích možností. V potaz se tentokrát nebere tuzemská nabídka, ale přímo využití platformy Microsoft Azure, která je poskytována celosvětově. Jedná se o službu umožňující rychle vytvářet, nasazovat, škálovat a spravovat aplikace v rámci globální sítě datacenter společnosti Microsoft, od čehož se odvíjí cena. Platí se za využívání IT zdrojů, ne za jejich

vlastnění, těžko se tedy dopředu odhadují náklady. Ty porostou s tím, jak se bude výsledný produkt, který na Azure poběží, využívat[7].

Jelikož se jedná o poměrně novou technologii, není tolik rozšířena, což může být problém pro její dlouhodobější správu. Dalším omezením je přesné definování služeb, které může zákazník využívat a omezení jejich konfigurace.

4.5.7 Vyhodnocení

Po seznámení se s jednotlivými možnostmi nasazení byla vytvořena tabulka 8, která jednotlivé možnosti porovnává podle toho, jak splňují či nesplňují následující tři kritéria:

1. **Náklady** – vychází z výše uvedených přibližných nákladů na pořízení a běh. Do ceny se nezapočítává vlastní administrace.
2. **Variabilita** – rozumí se volnost administrátora nakládat s daným systémem (instalace dodatečných aplikací, vlastní správa softwaru, atd.).
3. **Škálovatelnost** – možnost rozšíření HW konfigurace a navýšení tak celkového výkonu.

V tabulce 8 jsou zelenou barvou označeny služby, které dané kritérium zcela plní, žlutě ty, které ho plní pouze částečně a červeně, které ho neplní.

	náklady	**variabilita**	**škálovatelnost**
Webhosting			
VPS			
Dedikovaný server			
Housing			
Cloud			

Tabulka 8: Srovnání jednotlivých možností nasazení.

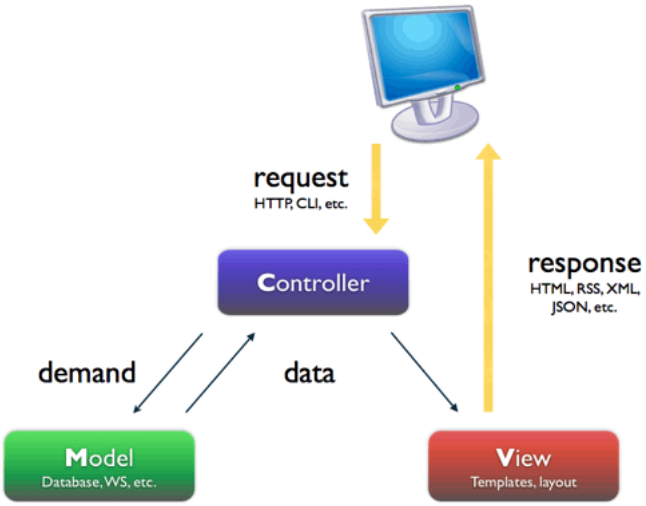
Podle získaných poznatků nejlépe vychází využití služeb VPS nebo Cloud pro produkční prostředí. Výsledky byly prezentovány zákazníkovi, který je vzal na vědomí.

[7]Výpočet orientační ceny je možné provést na `http://azure.microsoft.com/en-us/pricing/calculator`.

5 Analýza řešení s využitím platformy ASP.NET MVC

Pro vývoj byla zvolena technologie ASP.NET MVC 4, která implementuje návrhový vzor MVC a rozděluje tak projekt do tří základních částí. Následující kapitola analyzuje požadavky zadavatele a navrhuje jejich řešení s užitím zvolené technologie.

Na obrázku 7 je názorně znázorněna komunikace mezi jednotlivými částmi.



Obrázek 7: Návrhový vzor MVC.

Jelikož bylo zvoleno řešení s využitím platformy ASP.NET MVC, vymezují se možné použité programovací jazyky na C# a Visual Basic .NET. Vývojová prostředí se omezují na Microsoft Visual Studio, SharpDevelop či MonoDevelop případně i další, která ale nejsou obecně příliš zavedená, přičemž nativní podpora ASP.NET MVC verze 4 je přítomna pouze v Microsoft Visual Studiu.

5.1 Datová vrstva

Datová vrstva se stará o přístup k datům, která se nacházejí v datovém úložišti, typicky relační databázi nebo souborovém systému. V návrhovém vzoru MVC se práce na této úrovni řeší na úrovní Modelu, jehož jediným úkolem je definovat strukturu dat (pomocí datových kontejnerů) a tato data si získat z datového úložiště. Vrstva se žádným způsobem nestará o business logiku.

Jako datové úložiště může posloužit libovolná relační databáze zmíněná v sekci 3.2. Vzhledem k využití vývojových nástrojů Microsoftu se nabízí využití databáze

SQL Server, která je pro spolupráci s danými nástroji optimalizována. Ať už jde o nativní podporu objektově relačního mapování (ORM), tak pokročilé spravování rovnou v rámci Visual Studia.

Rozdíly mezi jednotlivými verzemi:

- SQL Server Express
  - Zdarma i pro komerční použití.
  - Omezení 10 GB na databázi.
  - Pouze jeden fyzický procesor (ale libovolně jader).
- SQL Server Standard
  - Není zdarma.
  - Bez výrazných omezení.
  - Umožňuje navíc analytické, integrační a notifikační služby.

V případě využití ORM se dá k vytvoření databáze přistoupit jednou ze tří metod:

Database First

Jak již název napovídá, databáze vzniká zcela samostatně. Po jejím vytvoření je podle jejího schématu reverzně vytvořen odpovídající model. V případě změny schématu databáze je nutné model buďto ručně upravit nebo ho přegenerovat.

Code First

V tomto případě jsou nejprve vytvořeny modelové třídy, které reprezentují jednotlivé entity a vazby mezi nimi. Dodatečné informace (primární klíč, maximální délka, apod.) se určují pomocí příslušného atributu, viz výpis 1.

Databáze je následně vytvořena podle těchto modelových tříd. V případě změny struktury během vývoje je možné využít tzv. migrací, kde se tyto změny nadefinují a databázi je možné za běhu upravit.

Model First

V tomto případě je nejprve vytvořen relační model, z kterého se vygenerují jak modelové třídy, tak SQL skript, který generuje databázi. V případě změn je nutné buď celou databázi přegenerovat podle nového SQL skriptu nebo ručně vytvářet tzv. change scripty, které databázi náležitě upraví.

```
public class Uzivatel
{
   [Key]
   public string Prezdivka { get; set; }
   public string CeleJmeno { get; set; }

   // virtuální vazba 1:N (1 uživatel může mít N příspěvků)
   public DbSet<Prispevek> Prispevky { get; set; }
}
```

Výpis 1: Ukázka třídy modelu.

5.1.1 Objektově relační mapování

Pro ORM je možné v .NET využít dvou frameworků, které nabízejí velmi podobnou funkcionalitu – NHibernate a Entity Framework. Mezi hlavní rozdíly by se dala zařadit poměrně neaktuální a necentralizovaná dokumentace NHibernate oproti přehledně udržované dokumentaci Entity Frameworku.[8]

5.1.2 Přístup k datům

Pro zajištění komplexního přístupu k datům slouží v *.NET Frameworku* objektový model *ADO.NET*.

ADO.NET je již celkem zaběhlá knihovna datových objektů, která dodává dva jmenné prostory databázového klienta – jeden je určen pro aplikaci *SQL Server* a druhý pro databáze používané prostřednictvím rozhraní *OLE DB*.

LINQ to SQL je integrovaný jazyk pro dotazování, který je součástí *ADO.NET* od verze *.NET Framework 3.5*. Jeho integrace je znázorněna na obrázku 8. Díky tomuto modelu je možné míchat SQL příkazy přímo s LINQ to SQL kódem, což usnadňuje případnou migraci stávajícího řešení. Ve výpisu 1 je ukázáno, jak se s ním pracuje.

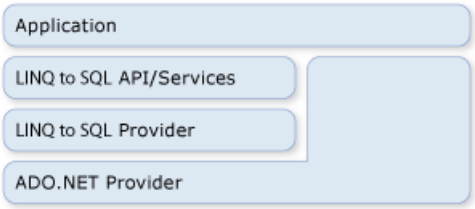
```
1 string[] names = { "Jan", "Hanka", "Evžen", "Iva", "Petr" };
2 var shortNames = from n in names
3                  where n.Length < 4
4                  select n
5                  orderby c.Length;
```

Výpis 2: Ukázka práce s *LINQ to SQL*.

[8] Velmi pěkné detailní srovnání lze nalézt v článku Daria Kucinskase [13].

Tímto kódem je zajištěno vytvoření proměnné `shortNames` (klíčové slovo `var` říká kompilátoru, aby si sám odvodil typ deklarované proměnné na základě kontextu – v tomto případě se jedná o typ *IOrderedEnumerable<T>*), která se naplní všemi jmény z pole `names`, jejichž délka je kratší nebo rovna čtyřem. Na konci dojte k seřazení podle délky. Dále se s těmito daty dá pracovat běžnými způsoby (např. použitím cyklu *foreach*).

LINQ jako takový je navržen hodně obecně. Výhodou je, že stejnou konstrukcí se dá pracovat s různými daty (např. vyhledávání dat v databázích, XML souborech či na filesystému může být zapsáno takřka stejným způsobem). Další výhodou může být objektový pohled na data.



Obrázek 8: Vztah ADO.NET a LINQ to SQL. [14]

5.1.3 Využití IoC/DI

Vzhledem k požadavkům na případnou rozšířitelnost aplikace by bylo vhodné využít nějakého z Inversion of Control a Dependency Injection (IoC/DI) frameworku. Mezi nejznámější patří SimpleInjector a Ninject.

5.1.4 Jazykové mutace

Jelikož je výsledný systém určený pro různé trhy v rámci EU, je potřeba aby bylo jeho prostředí přepínatelné do různých jazyků, ale aby data vedená u produktů/zboží byla rovněž v různých zemích různá (cena, popis produktu, ...).

Na základě požadavků od klienta se předpokládá, že každý produkt bude obsahovat:

- Jazykově závislá data
  - Název, prodejní cena, popis, různé soubory (manuály).
- Jazykově nezávislá data
  - Číslo produktu, čárový kód, informace o počtu kusů v balení, váha, ostatní.

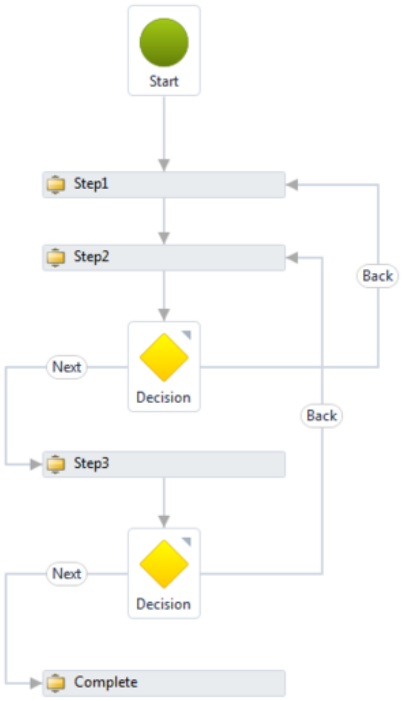
K tomuto problému se dá přistupovat dvěma základními způsoby – „jazykově závislá data“ budou individuální pro každý stát nebo budou individuální pro každý jazyk. V prvním případě je nutné mít na paměti možnou redundanci dat, jelikož mohou vznikat dva překlady téhož. V druhém případě může nastat problém s prodejní cenou, kterou může klient vyžadovat v různých regionech rozdílnou.

5.2 Aplikační vrstva

V návrhovém vzoru MVC se tato vrstva řeší na úrovni Controller. Jejím úkolem je přijímat požadavky od uživatele a řešit vnitřní chování aplikace, neboli business logiku. Pracuje jak s daty (model), tak s prezentační vrstvou (view) a vytváří mezi nimi spojení.

5.2.1 Workflow

V případě, že se řeší nějaká předdefinovaná komplexnější činnost, která se skládá z menších operací, stará se o běh tzv. workflow.



Obrázek 9: Workflow ve WF.

Příkladem jednoduchého workflow v tomto případě může být např. odeslání zakoupeného zboží, které se skládá z činností:

- **Na straně vlastníka zboží** – výběr zboží k odeslání, zvolení zákazníka, odeslání požadavku na expedici zboží.
- **Na straně skladu** – přijetí požadavku na expedici zboží, zabalení, potvrzení expedice zboží.

Workflow lze řešit buď individuálně nebo např. s využitím Windows Workflow Foundation (WF), který je součástí .NET Frameworku od verze 3.

Individuální přístup je vhodnější v případě projektů s menším počtem komplexních činností.

Využití WF se nabízí u projektů s rozsáhlejšími, popsanými procesy, u kterých dochází k návratu k předchozím stavům a tím pádem k cyklickým operacím. WF se implementuje jako samostatný projekt, který prostřednictvím

API zpřístupňuje a umožňuje práci s nadefinovanými workflow, které se definují v XAML souborech. Příklad obecného workflow ve WF je vidět na obrázku 9. V rámci tohoto deterministického stavového automatu je možné vykonávat akce jak na přechodech, tak ve stavech samotných.

Výhodou WF je jeho variabilita (dají se v něm dělat velmi sofistikované akce), škálovatelnost, nemusí se psát příliš kódu, díky oddělení od zbytku projektu je možné pro něj vytvářet samostatné automatické testy. Nevýhodou je, že potřebuje pro běh vlastní databázi, ukládaná data jsou v nečitelné podobě (nejde za běhu modifikovat) a příliš se nepoužívá.

5.2.2 Autentizace

Pro přístup do systému je nutné se přihlásit (autentizovat), žádná část systému kromě přihlašovací obrazovky a stránky s informacemi nemá být přístupná nepřihlášenému uživateli.

Dle zvyklostí internetových systémů se uživatel může autentizovat buď svým uživatelským jménem nebo emailem a heslem. Hesla je vhodné z bezpečnostních důvodů hashovat pomocí některého z vhodných algoritmů a ideálně ho před tím i tzv. ,,solit“.

Mezi nejpoužívanější algoritmy pro hashování se v dnešní době řadí MD5 a SHA1. Těm ale bývá vytýkána velká rychlost výpočtu otisku – díky rychlému hardwaru (např. i GPU) lze k prolomení takto uložených hesel využít metodu hrubé síly. Moderní alternativou jsou pomalejší hashovací algoritmy jako PBKDF2, SCrypt a BCrypt, které nemají rychlost vázanou na výkon HW, na kterém jsou počítány.

Přihlášený uživatel se může ověřovat buď pomocí autentizační cookie, která je šifrovaná, popř. pomocí Session. Aparát pro přihlašování je možné implementovat vlastní nebo se nabízí využití ASP.NET Membership providera a Role providera.

5.2.3 Autorizace

Každý z uživatelů systému má přidělenou jednu z definovaných rolí v části 4.3.2 – správce, vrchní skladník, skladník a vlastník zboží. Každá role vyžaduje přístup k jiné části aplikace s výjimkou vrchního skladníka, který má pouze drobně rozšířené pravomoci ve srovnání s běžným skladníkem. Nabízí se tedy rozdělení controllerů do tří základních částí podle rolí ,,správce“, ,,skladník a vrchní skladník“, ,,vlastník zboží“ a jedné společné části, která by řešila obecnou práci se všemi uživatelskými účty, zpracovávala chyby atd.

Pro kontrolu rolí slouží v ASP.NET MVC tzv. autorizační atribut, který se váže

buď k celému controlleru nebo konkrétní akci, přičemž větší prioritu má atribut uvedený u konkrétní akce. Příklad autorizačního atributu, který kontroluje, zda je uživatel přihlášený a jeho role je skladník nebo vrchní skladník je vidět ve výpisu 3.

```
1 [Authorize(Roles = "Skladnik, Vrchni_skladnik")]
2 public ActionResult MojeAction()
3 {
4    // vykonávaný kód
5 }
```

Výpis 3: Příklad autorizačního atributu.

V případě zvláštních požadavků na chování, např. přesměrování na individuální výsledky při nesplnění požadovaných předpokladů, je možné vytvořit si vlastní autorizační atribut, který bude stávající upravovat nebo rozšiřovat.

5.2.4 API

Aplikační rozhraní, které bude zpřístupňovat obsah systému, může být řešeno např. prostřednictvím REST (Representational State Transfer) nebo SOAP (Simple Object Access Protocol). Obě služby nejčastěji využívají standardní internetový protokol HTTP. SOAP se vyznačuje striktnějším zápisem a nutností vytvoření WSDL (Web Services Description Language) souboru, kde se uvádí možné metody a jejich parametry používané při komunikaci.

Jako odpověď se posílá nejčastěji požadovaný serializovaný objekt, buď ve formátu XML, JSON nebo jako prostý text, přičemž je nutné jeho strukturu definovat v dokumentaci.

Pro vytvoření API založeného na RESTu se v ASP.NET MVC využívá rozložení projektu označené jako WEB API. V případě SOAP je nutné využít WCF (Windows Communication Foundation) a vytvořit službu ideálně jako samostatnou knihovnu.

5.3 Prezentační vrstva

V návrhovém vzoru MVC se tato vrstva řeší na úrovni View. Definuje se zde webová stránka, která se posílá klientovi jako odpověď. Výsledná stránka se skládá z textu, odkazů na obrázky, občas i na jiná média. Rozložení, typografické a barevné schéma se řeší v kaskádových stylech (CSS). Dynamické chování zajišťuje javascript, buď client-side nebo server-side pomocí asynchronních dotazů AJAX (Asynchronous JavaScript and XML).

ASP.NET MVC narozdíl od klasického ASP.NET nedefinuje žádné kontrolky a vzhled se musí řešit zcela od základu pomocí běžných HTML tagů a CSS.

Existují nástroje jako Telerik[9] nebo DevExpress[10], které přidávají snadno nastavitelné multifunkční komponenty, jako např. gridy, grafy, kalendáře, wysiwyg editory, atd., které fungují jako celek a odstíní tak programátora od nutnosti psát vlastní scripty či styly. Nevýhodou je, že jsou tyto nástroje placené.

5.3.1 View engine

View engine je odpovědný za vytváření webových stránek z tzv. view. View obsahují jak HTML značky, tak programový kód, který se před odesláním ze serveru kompiluje.

ASP.NET používá vlastní view engine nazývaný WebForm, kde se programový kód zapisuje mezi tagy `<% %>` a `<%: %>`.

S příchodem ASP.NET MVC3 přibyl do základního výběru další view engine Razor, který nabízí čitelnější syntaxi: `<div>@Model.PrihlasovaciJmeno</div>`, práci přímo s ViewModely a obecně lépe respektuje architekturu MVC.

Kromě základních dvou enginů existuje ještě celá řada alternativ, jako SharpTiles (částečně založený na JSTL z Javy), Spark View Engine (snaha splynout s HTML značkami pro snazší čitelnost), NHaml, NDjango, Hasic, Brail, atd.

5.3.2 Razor rozložení

Webová stránka se může skládat z následujících tří částí:

- **Layout**
  - Definuje celkové rozložení stránky – tradičně výběrové menu, hlavičku, obsahovou část, patičku, atd.
  - Bývá pro všechny stránky stejný, mění se pouze obsah, toho se docílí příkazem `@RenderBody()`, zde je poté umístěno view vrácené controllerem.
  - Může definovat vlastní sekce pomocí `@RenderSection()`, které poté lze ve view modifikovat (typicky linkování CSS nebo javascritu určené pro konkrétní stránku, změna textu v menu, informačním proužku apod.).

[9] `http://www.telerik.com/aspnet-mvc`
[10] `https://www.devexpress.com/Products/NET/Controls/ASP/MVC`

- **View**
  - Obsahuje informaci o zvoleném layoutu, který se má použít.
  - Bývá svázána s ViewModelem, odkud se berou data.
  - Reprezentuje část s obsahem.
  - Může obsahovat tzv. partial view, které reprezentují nějakou komponentu (použití pomocí `@Html.Partial("NazevPartialView", model)`).
- **Partial View**
  - Menší nedělitelný celek, bývá svázán s modelem, kde jsou umístěná data.
  - Musí existovat view, které ho zobrazuje.
  - Neumí přepisovat sekce (není možné přiřadit mu vlastní CSS nebo javascript soubory, ty se musí linkovat již ve view).
  - Používá se pro větší přehlednost a eliminaci stejného kódu v různých view.
  - Používá se pro práci s AJAXem.

5.3.3 Javascript

Jak již bylo zmíněno, javascript slouží pro zajištění dynamického chování webové stránky. Asi nejpopulárnější knihovnou pro práci s javascriptem je jQuery, která se automaticky integruje do jakéhokoliv ASP.NET MVC projektu. Využívá se zde hlavně pro validaci formulářových dat na straně klienta s využitím pluginu jQuery Validation. Tuto validaci je možné zajišťovat automaticky – generuje se na základě atributů uvedených u jednotlivých vlastností na ViewModelu. Výhodou při uvádění validačních atributů přímo na modelu je jejich snadná kontrola i na úrovni serveru, která je nezbytná, jelikož posílaná data mohou být vždy podvržena.

Výhodou jQuery je i značná četnost volně dostupných pluginů, které řeší rozličné činnosti jako např.:

- **jsTree** – http://www.jstree.com
  - Umožňuje vytváření interaktivních stromů, použitelné např. pro třídění produktů (tagy),
- **Select2** – http://ivaynberg.github.io/select2
  - Upravuje práci výběrových polí (HTML Select), nabízí vyhledávací pole, přehledný hromadný výběr a dynamické dotahování dat.

Dále jQuery zjednodušuje zápis asynchronních dotazů na server pomocí AJAX.

5.3.4 Kaskádové styly

Kaskádové styly je možné spravovat buďto rovnou, ale pro pokročilou práci s kaskádovými styly je možné využít některý z CSS preprocesorů, jako jsou Sass nebo LESS. Ty částečně řeší špatnou udržitelnost kaskádových stylů, které se v průběhu času modifikují a rozšiřují. Zavádí totiž do CSS některé vlastnosti z programovacích jazyků, jako jsou např. ,,proměnné“, ,,výrazy“, ,,podmínky“, umožňují se vzájemně importovat apod. Takto zapsané zdrojové kódy se na pozadí rovnou překládají do výsledných CSS souborů. [15] Nevýhodou může být horší podpora IntelliSense ve Visual Studiu.

Dále je při návrhu webů možné využít již hotových frontend frameworků, které zajišťují moderní responzivní vzhled, jako jsou Bootstrap či Zurb Foundation. Jde o sadu nástrojů usnadňující práci s typografií, tvorbu layoutu, vytváření elementů uživatelského rozhraní a zároveň ošetřující zobrazování napříč platformami. [16] Nevýhodou může být problematická modifikace různých stylů, pokud je vyžadován jiný než výchozí vzhled. Hodí se převážně při vývoji aplikací, které mají svůj vzhled měnit dle použitého zařízení (desktop, tablety, chytré telefony).

5.3.5 Bundling a minifikace

Soubory kaskádových stylů a javascriptu je možné k příslušným stránkám přilinkovat buď přímo nebo se nabízí využití tzv. bundlingu. Při něm dochází ke spojení jednotlivých – typicky na sobě závislých – souborů do jednoho virtuálního souboru s vlastní adresou, čímž se redukuje počet dotazů na server. To je velmi užitečné vzhledem k faktu, že většina dnešních nejpoužívanějších prohlížečů omezuje počet současných připojení k jednomu hostname na šest. [17] K sestavení bundlů dochází typicky pouze jednou a to při startu aplikace. Při každém novém sestavení je vygenerována pro jednotlivé bundly vlastní uri adresa, která se skládá jak z nadefinovaného jména, tak náhodného hashe, čímž je zajištěno, že prohlížeč nepoužije starou verzi scriptu/stylu, kterou může mít v cache, protože se z jeho pohledu jedná o nový soubor.

Po bundlingu se provádí nad vytvořenými virtuálními souboru ještě minifikace, jejímž účelem je zmenšit odesílané soubory a celý proces komunikace s klientem tak urychlit.

5.3.6 Překlady prostředí

Pro tvorbu překladů prostředí se v .NET standardně využívá tzv. Resource souborů (.resx). Jedná se o XML soubory s pevně danou strukturou, která v sobě může uchovávat informace v různých formátech (texty, obrázky, ikony, audio, obecně soubory a ostatní). Pro případ textu je ve výpisu 4 naznačen příklad zápisu. Skládá

se ze tří základních částí – unikátního jména, které slouží pro rozlišení záznamů („ErrorMessage“), hodnoty a volitelného komentáře.

```
1 <data name="ErrorMessage" xml:space="preserve">
2    <value>Přihlašovací jméno je povinný údaj.</value>
3    <comment>Chybová hláška</comment>
4 </data>
```

Výpis 4: Ukázka záznamu v .resx souboru.

V praxi se nejprve se vytvoří základní .resx soubor s výchozími texty, obrázky, atd. Přístup k těmto záznamům je v kódu dostupný pomocí třídy `ResourceManager`. Pro překlady do jiných jazyků se vytváří vlastní .resx soubory, které se jmenují stejně, ale před příponou obsahují informaci o zvoleném jazyku a kultuře. Pro anglický překlad se tedy vytvoří .en.resx, pro slovenský .sk.resx atp. Je možné specifikovat i konkrétní kulturu, takže pro americkou angličtinu se použije en-US.resx. V překladových Resource souborech se použijí stejné identifikátory a přeložené entity (texty, obrázky, ...).

Pro překlady je také možné využít některou z knihoven, např. FairlyLocal[11]. Nepříliš vhodnou alternativou může být i začlenění překladů přímo do zdrojových kódů programu.

5.4 Správa verzí

Pro správu verzí je možné využít TFS (Team Foundation Server), který je přímo integrován do vývojového prostředí, nebo instalace SVN (Apache Subversion). V druhém případě je možné spravovat verze zdrojového kódu mimo vývojové prostředí nebo využití některého z rozšíření pro integraci SVN přímo do Visual Studia.

V obou případech je nutné zřídit si server pro ukládání zálohovaných a verzovaných dat.

[11] `http://www.fairtutor.com/fairlylocal`

6 Návrh aplikace ,,virtuální sklady“

V této kapitole se uvádí návrh řešení aplikace ,,virtuální sklady“, pracovně pojmenované jako Vinry (vzniklo zkrácením anglického originálu Virtual Inventory).

6.1 Terminologie

Kromě níže uvedených uživatelských rolí se dále v textu používají následující termíny:

Produkt

Obecný produkt/výrobek, který reprezentuje zboží uskladněné ve virtuálním skladu. Předpokládají se ,,neproblematické produkty“, tudíž sem nepatří potraviny, chemikálie, zvířata, apod.

Jazyk prostředí

Jazyk webového prostředí – GUI (Graphical User Interface).

Jazyk obsahu

Jazykově/lokačně závislé informace o produktu, jako je například cena, popis produktu aj. Nejedná se o jazyk webového prostředí.

Objednávka

Seznam zboží vč. jeho množství, který sestavil vlastník zboží a které má být zabaleno a odesláno zvolenou expediční službou konečnému zákazníkovi.

Dodávka

Seznam zboží vč. jeho množství, který sestavil vlastník zboží a které má být ve skladu očekáváno pro příjem a naskladnění.

6.2 Rozdělení uživatelů podle rolí

Vlastník zboží

Vlastníkem zboží je myšlen subjekt (typicky organizace), pod kterou může spadat jeden nebo i více uživatelů. Každý takový uživatel má stejná práva a může provádět stejné akce. Jak již název napovídá, jedná se o právoplatného vlastníka zboží, který využívá virtuální sklady pro uskladnění svého zboží.

Základní akce:

- Vytváření nových produktových listů (ručně či automatizovaný import).
- Přikládání globálních i lokálních informací o produktu, připojování obrázků a souborů.
- Vytváření štítků a přiřazení štítků k produktům.
- Vytváření dodávek pro příjem zboží do skladů.
- Vytváření objednávek pro odeslání zboží ke koncovému zákazníkovi.
- Zobrazení osobních reportů.

Správce

Jedná se o administrátora celého systému. Je pouze jeden.

Základní akce:

- Přidávání nových skladů, vlastníků zboží, skladníků a jmenování vrchních skladníků.
- Zobrazení všech reportů.
- Zobrazení logů.

Skladník

Zaměstnanec skladu, který fyzicky manipuluje se zbožím. Má přístup pouze k informacím, které jsou pro něj nezbytné – jako jsou produkty vlastníka zboží, které se vážou k danému skladu (bez cen); dodávky a objednávky.

Základní akce:

- Příjem zboží na základě dodacích listů.
- Reklamace počtu kusů přijímaného zboží.
- Odesílání zboží konečnému zákazníkovi na základě objednávek od vlastníka zboží (vidí součet cen kvůli pojištění).
- Zobrazení všech vlastníků zboží, jejich produktů a zákazníků evidovaných k danému skladu.

Vrchní skladník

Jedná se o skladníka s rozšířenými pravomocemi – může přidávat nové skladníky pro svůj sklad, má přístup k citlivějším údajům jako jsou např. ceny produktů.

6.3 Vlastník zboží

Popis dílčích akcí uživatele s právy ,,vlastník zboží“.

6.3.1 Nastavení

V sekci nastavení může měnit své osobní údaje, jako je heslo, výchozí jazyk a počet záznamů na stránku i nastavení celého subjektu, jako je číslo účtu a hodnota DPH. Zároveň se zde nachází správa uživatelů daného subjektu, které je možné přidávat nebo naopak mazat. Není povoleno smazat svůj vlastní účet.

6.3.2 Produkty

V sekci produkty je umožněno si procházet a filtrovat existující produkty. Je možné si přidávat nové, zobrazovat a editovat stávající. K produktu mohou být připojeny obrázky i soubory. Smazání produktu je povoleno pouze pokud se s ním žádným způsobem nepracovalo, tedy pokud není součástí žádné objednávky ani dodávky.

6.3.3 Zákazníci

V sekci zákazníci je umožněno si procházet a filtrovat existující zákazníky. Je možné si přidávat nové, zobrazovat a editovat stávající. U zákazníka lze evidovat jednu nebo více adres. Smazání zákazníka je povoleno pouze pokud se s ním žádným způsobem nenakládalo – nesmí být součástí žádné objednávky.

6.3.4 Objednávky

V sekci objednávky je umožněno si procházet a filtrovat existující objednávky. Je možné vytvářet nové a zobrazovat stávající. Objednávky není nutné ihned po vytvoření odeslat, je umožněno udržovat jejich koncept – ten se dá editovat, mazat nebo odeslat. Po odeslání již nelze objednávky editovat ani mazat. Při sestavování objednávky je nutné nejprve zvolit sklad, poté až seznam produktů, jejich počet, adresu koncového zákazníka a požadovanou spediční službu. V případě změny skladu jsou všechny údaje smazány a je nutné je navolit znovu.

Objednávka si prochází následujícími stavy:

- **Koncept** – rozpracována vlastníkem zboží, je možné ji editovat,
- **odeslaná** – objednávka se nyní zobrazuje skladníkům daného skladu, je možné ji editovat,
- **přijata** – nějaký skladník se jí ujal, je možné ji stornovat,
- **stornovaná** – volitelný koncový stav,
- **zpracovaná** – skladník připravil balík/y k odeslání,
- **vydaná** – expediční služba převzala balík/y,
- **uzavřená** – koncový stav, vlastník zboží označil objednávku za dokončenou.

6.3.5 Dodací listy

Pro dodací listy platí stejný přístup jako pro objednávky s výjimkou určování zákazníka. Zboží v tomto případě přichází na sklad a ne obráceně.

Dodací list si prochází následujícími stavy:

- Koncept – rozpracovaný vlastníkem zboží, je možné ho editovat,
- odeslaný – dodací list se nyní zobrazuje skladníkům daného skladu, je možné ho editovat,
- přijatý – nějaký skladník se ho ujal, editace již není možná,
- reklamovaný – volitelný stav; dle skladníka nesouhlasí počty kusů, vlastník zboží je informován o počtu kusů, ke kterým došel skladník; tento stav trvá dokud vlastník neodsouhlasí počet kusů zadaný skladníkem. V případě rozporů je nutné problém řešit mimo systém. Skladník může reklamovaný počet kusů editovat, dokud ten není schválen vlastníkem zboží.

- uzavřený – koncový stav, může nastat dvěma způsoby:
  1. Skladník nenahlásil reklamaci a označil dodací list za uzavřený,
  2. Skladník nahlásil reklamaci a list musí uzavřít vlastník zboží.

6.3.6 Reporty

Sekce reporty je pouze pro čtení – formou tabulek jsou zde zobrazeny transakce za jednotlivé měsíce rozdělené podle skladů. Při kliknutí na konkrétní transakci je uživateli zobrazen detail podle typu transakce (objednávka nebo dodávka). Každá tabulka obsahuje i sumarizační řádek.

Dále je zde ke stažení vyúčtování za služby spojené s vedením jednotlivých skladů, které sem nahrává správce systému jako soubory formátu PDF.

6.4 Skladník

Popis jednotlivých akcí uživatele s právy skladníka. Skladníkovi se nezobrazují ceny a to ani v případě, že má zobrazen stejný výpis jako vlastník zboží, např. detail produktu, objednávky, dodávky atd. Jediný rozdíl je zobrazení celkové ceny balíků objednávek, kterou je nutné znát.

6.4.1 Dodací listy

V sekci dodací listy je zobrazen seznam existujících dodacích listů v daném skladu. Dodací listy je možné filtrovat podle stavů popsaných výše v sekci 6.3.5 (pouze koncepty nejsou zobrazovány) a zároveň si může skladník nechat zobrazit pouze dodací listy, kterých se ujal – tedy takové, s kterými může pracovat. Každý skladník daného skladu si může zobrazit libovolný dodací list pro čtení.

V případě, že není dodací list nikomu přiřazen, může se ho skladník ujmout. Tato akce je nevratná, již přiřazenému dodacímu listu není možné změnit skladníka. Skladník zodpovídá za převzetí zboží a jeho kontrolu. V případě nesouladu zadá skladník do systému reklamaci, kterou musí vlastník zboží schválit.

6.4.2 Objednávky

V sekcí objednávky je zobrazen seznam existujících objednávek v daném skladu. Objednávky je možné filtrovat podle stavů popsaných výše v sekci 6.3.4 (pouze

koncepty nejsou zobrazovány) a zároveň si může skladník nechat zobrazit pouze objednávky, kterých se ujal – tedy takové, s kterými může pracovat. Každý skladník daného skladu si může zobrazit libovolnou objednávku pro čtení.

V případě, že není objednávka nikomu přiřazena, může se jí skladník ujmout. Tato akce je nevratná, již přiřazené objednávce není možné změnit skladníka. Dokud neoznačí skladník objednávku jako zpracovanou, může ji vlastník zboží stornovat. S takovou objednávkou nelze nadále pracovat.

6.4.3 Nastavení

V sekci nastavení může měnit své osobní údaje, jako je heslo a počet záznamů zobrazených na stránku.

6.4.4 Subjekty

V sekci subjekty je seznam všech vlastníků zboží (nejsou zde zohledňováni konkrétní uživatelé, pouze celé subjekty). Celá sekce je určena pouze pro čtení. Po kliknutí na konkrétního vlastníka se zobrazí jeho detail se čtyřmi záložkami – výčet všech objednávek, dodávek, zákazníků a produktů, které se vážou k danému skladu.

6.5 Správce

6.5.1 Editace číselníků

V této sekci je možné zobrazit si a editovat hodnoty většiny tabulek typu číselník (viz sekce 6.7). Konkrétně se jedná o výčet jazyků, států a expedičních služeb dostupných konkrétním skladům. Není umožněno mazání hodnot, které se již někde používají.

6.5.2 Správa skladů

V této sekci je možné zobrazit si a editovat údaje o skladech (adresa a název). U detailu skladu je seznam skladníků a vlastníků zboží, kteří jsou s tímto skladem svázáni.

Nad skladníky lze provádět standardní CRUD akce (create, read, update, delete). V případě mazání nedochází k fyzickému mazání z databáze, ale k pouhému skrytí (zamknutí). K zamčenému účtu není možné se přihlásit, takový účet se chová jako neexistující. Kromě běžných údajů uvedených v 6.4.3 je možné přidělit či odebrat skladníkovi roli vrchního skladníka.

6.5.3 Správa subjektů

Nad vlastníky zboží lze rovněž provádět CRUD akce. V detailu subjektu je k dispozici jeho report, je zde možné nahrávat a mazat soubory (měsíční vyúčtování) a nachází se zde správa uživatelů daného subjektu s kterou jdou opět provádět CRUD akce.

6.5.4 Logy

V sekci logy se nachází jednoduchý sekvenční výpis akcí uživatelů. Eviduje se název uživatele, provedená akce, poznámka a čas. Jsou zde zaznamenávány i pády aplikace, přičemž do poznámky se ukládá zachycené chybové hlášení.

6.6 Struktura aplikace

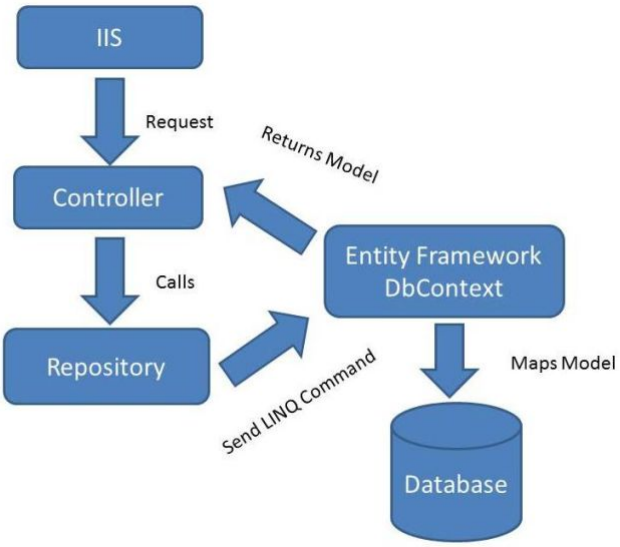
Aplikace byla rozdělena do tří logických celků, které jsou v rámci celého řešení (Solution) zastoupeny následujícími projekty:

1. `TT.Vinry.DAL`
   Tento projekt řeší přístup a práci s daty – DAL (data access layer). Obsahuje ORM vytvořený pomocí Entity Frameworku metodou database first. Jeho výstupem je samostatná knihovna, která může být použita i v jiných aplikacích.

2. `TT.Vinry.Importer`
   Jednoduchá WPF (Windows Presentation Foundation) aplikace, která umožňuje importovat data z vzorové dBase databáze (exportovaná z ekonomického informačního systému SQL Ekonom), který obsahuje seznam testovacích produktů. K přístupu k databázi využívá `TT.Vinry.DAL`.

3. `TT.Vinry.Web`
   Hlavní projekt webového systému – řeší se zde GUI, aplikační logika a pomocí knihovny `TT.Vinry.DAL` se zde pracuje s daty.

Zdrojové kódy i komentáře jsou psané v anglickém jazyce, jsou dodržovány zásady dle [18] pro štábní kulturu.



Obrázek 10: Použitý návrhový vzor Repository. [19]

Pro vývoj byl zvolený návrhový vzor Repository, který vytváří abstraktní vrstvu mezi datovou vrstvou a aplikační logikou. Princip je znázorněn na obrázku 10.

6.7 Model

Kvůli oproštění od množství rutinních úkonů při prací s databází bude použit ORM – konkrétně s využitím ADO.NET Entity Frameworku, který vygeneruje objektovou strukturu. Výhodou je kromě silného typování, IntelliSense a validace při kompilaci i automatické vytvoření tříd pro práci s daty – kontejnerů. Toho se využívá v projektu `TT.Vinry.DAL` i ve všech dalších projektech, které tuto knihovnu využívají.

Jelikož jsou pro view občas potřeba data s jinou strukturou, než která odpovídá tabulkám v databázi, definují se v projektu `TT.Vinry.Web` další třídy modelu, tzv. ViewModel třídy. Každá stránka nese informaci o tom, jaký ViewModel je jí od controlleru předáván, aby mohla s daty snáze nakládat.

6.7.1 Relační databázový model

V příloze B je znázorněn diagram navrhovaného relačního databázového modelu, který byl schválen zákazníkem. Při jeho vytváření se vycházelo z doporučení [20, 21] pro tvorbu efektivních datových modelů.

6.7.2 Popis tabulek

Pomocí odsazení je naznačeno logické zařazení jednotlivých tabulek kvůli snazší orientaci.

OWNER reprezentuje subjekt vlastníka zboží.

- **OWNERUSER** reprezentuje jednotlivé uživatele subjektu.
- **LANGUAGE** rozkladová tabulka mezi subjektem a všemi jazyky (určuje v jakých jazycích může být zadáván obsah).
  - **POSSIBLELANGUAGES** tabulka typu číselník, obsahuje výčet dostupných jazyků.
- **OWNERINVENTORY** rozkladová tabulka mezi vlastníkem zboží a sklady (určuje jaké sklady může vlastník využívat).
- **PRODUCT** reprezentuje všechny produkty vlastníka zboží.
  - **PRODUCTINVENTORY** rozkladová tabulka mezi produktem a skladem (určuje počet kusů daného produktu v daném skladu).
  - **PRODUCTIMAGE** reprezentuje obrázky vázané k danému produktu.
  - **PRODUCTDATA** reprezentuje jazykově závislá data produktu (název, cena, ...).
  - **PRODUCTFILE** reprezentuje jazykově závislé soubory vázané k produktu (manuály, letáky, ...).
  - **DELIVERYPRODUCT** rozkladová tabulka mezi dodávkou a produktem (určuje příchozí zboží na sklad).
  - **ORDERPRODUCT** rozkladová tabulka mezi objednávkou a produktem (určuje zboží, které má být odesláno zákazníkovi).
  - **PRODUCTTAG** rozkladová tabulka mezi produktem a tagem (určuje do jakých skupin produkt náleží).
    - **TAG** tabulka typu číselník, obsahuje výčet všech tagů.
- **CUSTOMER** tabulka pro zákazníky vlastníka zboží.
  - **DELIVERYADRESS** tabulka s adresami zákazníků.

INVENTORY reprezentuje virtuální sklady.

- **COUNTRY** tabulka typu číselník, reprezentuje státy pod které může sklad patřit.
- **STOREMAN** reprezentuje skladníky.
- **DELIVERYTYPE** tabulka typu číselník, určuje spediční služby dostupné danému skladu.
- **ORDER** reprezentuje objednávky odesílané konečnému zákazníkovi.

ORDERSTATE rozkladová tabulka mezi stavem a objednávkou (určuje stavy objednávek).

OSTATE tabulka typu číselník, obsahuje výčet stavů objednávek.

DELIVERY reprezentuje dodávky – produkty které mají být naskladněny.

DELIVERYSTATE rozkladová tabulka mezi stavem a dodávkou (určuje stavy dodávek).

DSTATE tabulka typu číselník, obsahuje výčet stavů dodávek.

6.7.3 Přístupová práva

Protože mají jednotlivé uživatelské role značně rozdílné možnosti nakládání se systémem, jsou v databázi evidovány na různých místech. Neexistuje tedy žádná samostatná tabulka se všemi uživateli.

Vlastník zboží je evidován v tabulce OWNER, resp. OWNERUSER, zatímco skladník v tabulce STOREMAN. Jelikož je správce již v návrhu pouze jeden, není evidován v databázi, ale v konfiguračním souboru web.config, viz výpis 5.

```
1 <configuration>
2   <appSettings>
3     <add key="ADMIN_LOGIN" value="spravce" />
4     <add key="ADMIN_PASS" value="97b36a001fb3b6b657405ed2e22b52a0" />
5   </appSettings>
6 </configuration>
```

Výpis 5: Zadání přihlašovacích údajů ve web.config.

Hesla všech uživatelů se kryptují pomocí algoritmu PBKDF2.

6.7.4 Validace dat

Pokud uživatel posílá přes webový formulář nějaká data, je tato data nutné validovat – ideálně na straně klienta pomocí javascriptu, pokaždé však na straně serveru. Musí se počítat s tím, že uživatel může být útočník a přijímaná data mohou být úmyslně upravena. Pro vyřešení validace na obou stranách zároveň se využívá validačních atributů v kombinaci s view enginem Razor. Formuláře vytváří, jak je uvedeno ve výpisu 6.

Nejprve se vytvoří blok, kde se definuje jaká akce (`EditProduct`) v jakém controlleru (`Products`) se má volat jakým způsobem (POST). Na základě toho se vygeneruje formulář s příslušnou uri adresou dle přednastaveného routování.

```
@using (Html.BeginForm("EditProduct", "Products", FormMethod.Post))
{
   <!-- ... HTML ... -->

   @Html.VinryLabelFor(m => m.ProductNumber):
   @Html.TextBoxFor(model => model.ProductNumber)

   <!-- ... HTML a ostatní prvky formuláře ... -->

   <input type="submit" value="Uložit změny" />
}
```

Výpis 6: Ukázka práce s Razor enginem.

Formulář může obsahovat libovolné HTML značky pro zajištění vzhledu a struktury výsledné webové stránky, typicky se jedná o div bloky nebo tabulku v kombinaci s CSS.

```
[Required]     // validační atribut
[Display(Name = "Číslo produktu")]  // zobrazovací atribut
[MaxLength(20, ErrorMessage = "Maximální délka čísla produktu je {1}.")] //
    validační atribut
public string ProductNumber { get; set;}   // vlastnost
```

Výpis 7: Použití validačních a zobrazovacích atributů u ViewModelu.

Jelikož má stránka definovaný ViewModel s kterým může pracovat, je možné ho využít pro vytvoření jednotlivých prvků formuláře a při odeslání bude na controller poslán naplněný objekt. U ViewModelu lze pro jednotlivé vlastnosti využít jak atributů nejen pro validaci, ale i pro zobrazení.

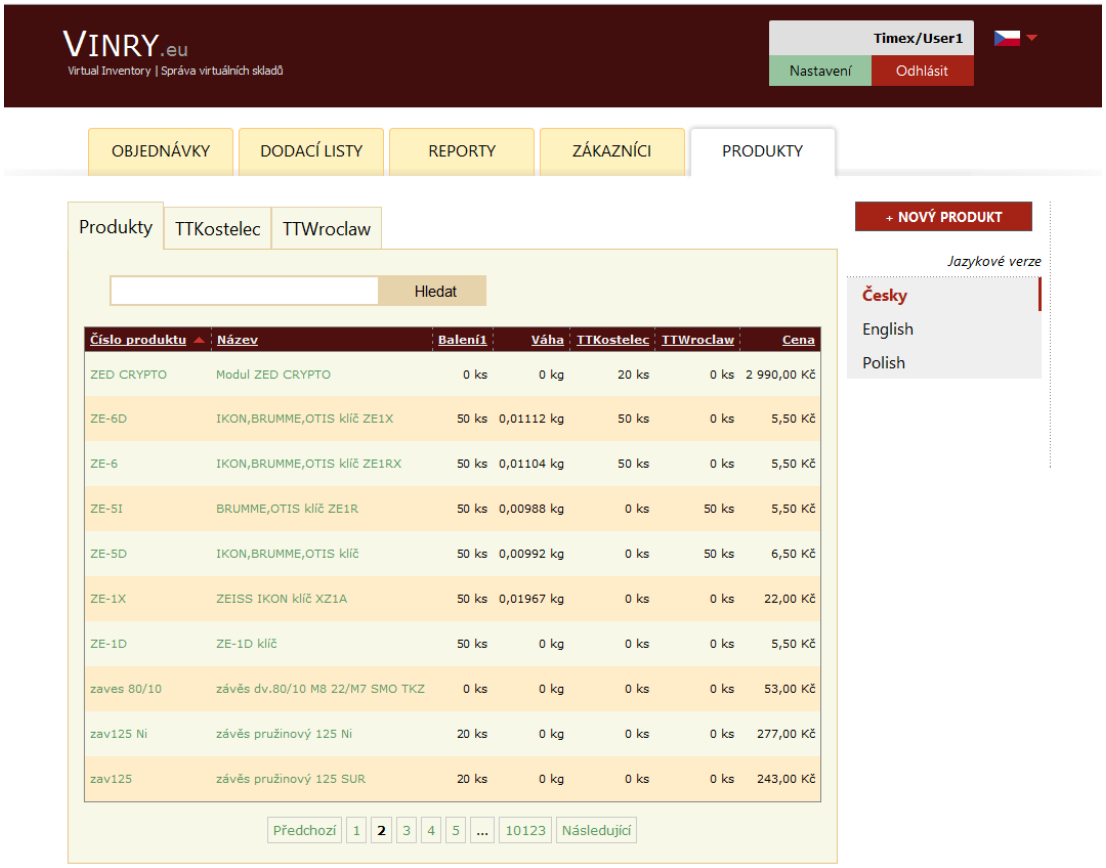
6.8 View

Jelikož byl požadován jednoduchý moderní vzhled s intuitivním ovládáním, byl na základě domluvy nejprve vytvořen tzv. prototyp webu, který se skládal z prostého HTML a obrázků a který navrhované GUI prezentoval. Ten byl předložen ke schválení zadavatelem.

6.8.1 Schválený grafický návrh

Na obrázku 11 je vidět schválená podoba grafického návrhu z pohledu vlastníka zboží. V hlavičce je vlevo vidět textové pracovní logo, které se bude při nasazení na produkci aktualizovat, vpravo je vidět ,,uživatelský box“, kde je umístěn název vlastníka, alias subjektu (Timex), jméno uživatele (User1) a tlačítka pro změnu

nastavení a odhlášení, a přepínač jazyků pro prostředí. V menu pod hlavičkou jsou vidět jednotlivé sekce, do kterých má uživatel přístup.



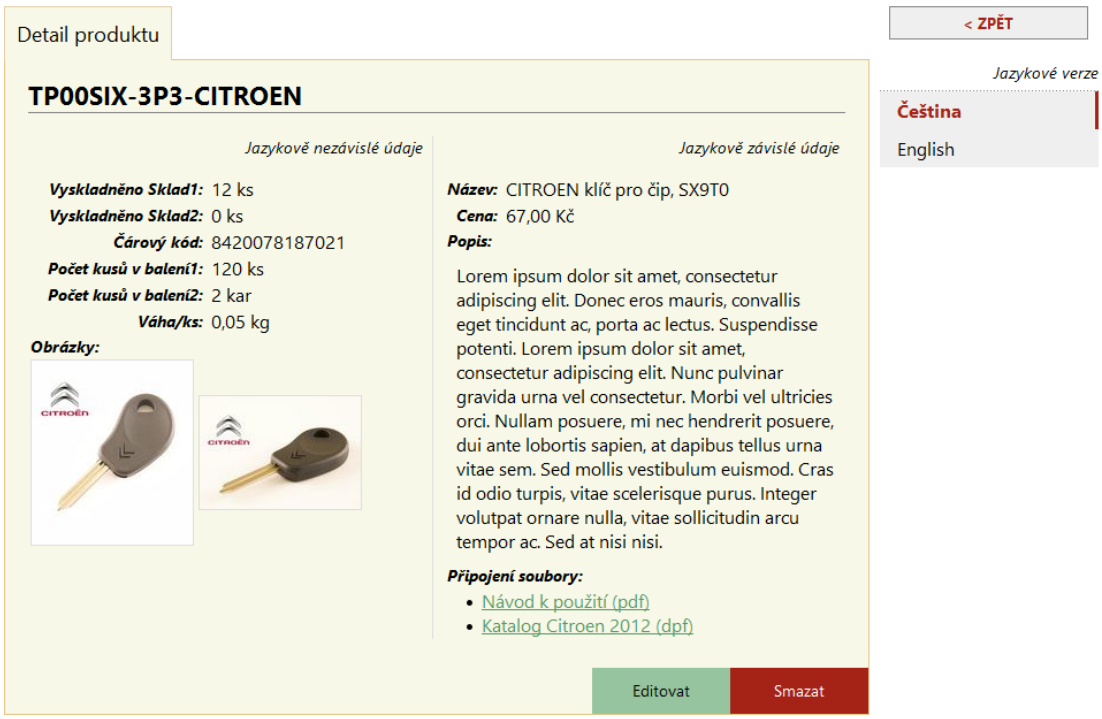
Obrázek 11: Schválený vzhled GUI.

V rámci produktů je uživateli zobrazena záložka všech produktů nebo je možné omezit pohled pouze na produkty v konkrétních skladech. Vpravo se nachází tlačítko pro standardní akci, v tomto případě přidání nového produktového listu. Dále je zde menu pro výběr jazyka obsahu.

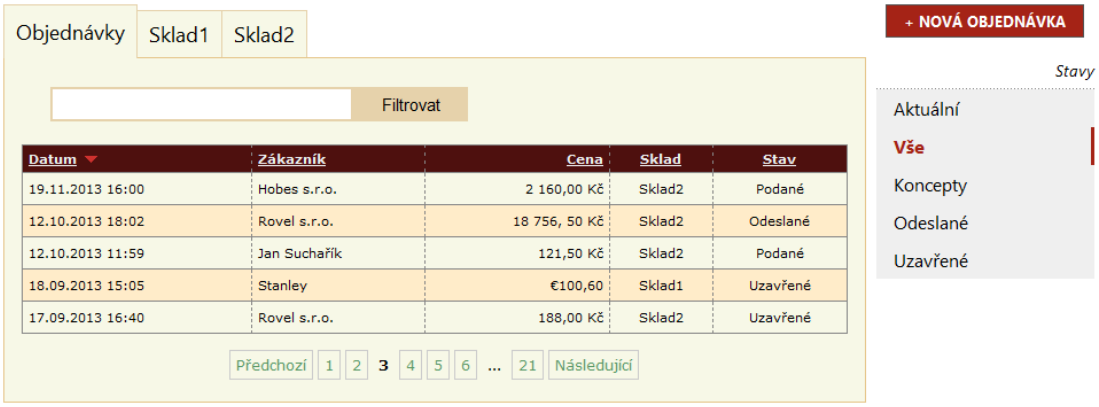
Jednotlivé produkty jsou zobrazeny v tabulce, která umožňuje vyhledávání dle názvu nebo čísla produktu, což byl požadavek zadavatele. Výsledky jsou stránkované a lze je řadit podle libovolného sloupce. Po kliknutí na produkt je zobrazen jeho detail. Ukázka detailu je vidět na obrázku 12.

Detail produktu je vertikálně rozdělen na dvě logické části – jazykově závislá a jazykově nezávislá data. Jazykově nezávislá data jsou pro všechny výskyty daného produktu stejná, jazykově závislá data se mění podle zvolené jazykové verze. Důvodem je využití tohoto systému v rámci různých zemí a případné budoucí napojení na e-shop vedený pro danou zemi. Mezi jazykově závislé údaje se počítá i cena, která ale ve skutečnosti není závislá na jazyku, ale na konkrétním státu.

To je ovšem možné zohlednit přímo při definování jazyků, kdy např. pro němčinu budou existovat dvě jazykové verze: ,,Deutsch (DE)“ pro Německo a ,,Deutsch (AT)“ pro Rakousko.



Obrázek 12: Ukázka detailu produktu.



Obrázek 13: Objednávky.

6.8.2 Layout

Pro všechny view se vytvoří jednotná sdílená _Layout.cshtml stránka, která bude respektovat schválenou podobu. Na úrovni layoutu se bude řešit navigace po systé-

mu a budou se zde definovat sekce pro vkládání CSS a javascriptu, na nichž jsou závislá konkrétní view.

6.9 Controller

Nepočítá se s velkou zátěží systému, tedy velkým počtem uživatelů, kteří by pracovali ve stejnou chvíli (řádově desítky, maximálně stovky). Je tedy možné ověřovat uživatele při každém dotazu na server a zajistit tak případně jeho odhlášení, pokud mu byla odebrána práva. Toho se dá docílit vlastním přepsáním autorizačního atributu, který je u přihlášeného uživatele stejně pokaždé vyžadován. V projektu tedy nebude používán výchozí autorizační atribut, ale jeho upravená verze, `VinryAuthorizeAttribute`.

6.9.1 Workflow

Jediné dvě části, u kterých se řeší přechody mezi stavy, jsou objednávky a dodávky. Dochází zde k jednoduchému větvení, přechody neobsahují cykly, tudíž bylo zvoleno řešení s využitím rozkladových tabulek, jak již je naznačeno v sekci věnované modelu (6.7).

6.9.2 Rozdělení controllerů

Každá třída uživatelů pracuje se systémem jinak, proto se controllery rozdělí na obecné, jako je `ErrorsController` nebo `BaseController`, které zaštiťují obecnou funkcionalitu a na controllery podle rolí, kde se řeší dílčí činnosti popsané výše, jako je `ProductsController`, `DeliveriesController` atd.

6.9.3 Zabezpečení přístupu

Jelikož se počítá s využitím vlastního autorizačního atributu, je ho možné využít nejen pro autentizaci, ale i autorizaci. Pokud má uživatel jinou roli, než jakou by měl mít, bude odhlášen a přesměrován na stránku pro nové přihlášení.

6.10 Řízení vývoje

Pro řízení vývoje byl zvolen TFS, který kromě správy zdrojových kódů umožňuje i řízení projektového vývoje. TFS bude nasazen na testovacím serveru a poslouží

primárně pro zálohování a verzování projektu. V případě problémů bude možné porovnávat případně se vrátit k libovolné verzi. Při každém odesílání bude vyžadováno zadání stručného komentáře pro evidenci provedených změn.

Projekt byl rozdělen do následujících větví:

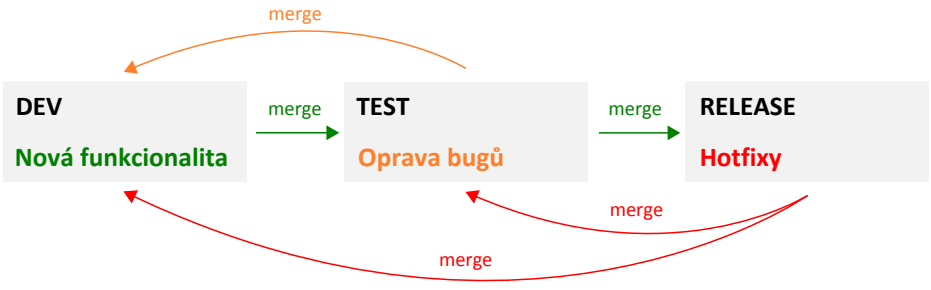
- **Dev** – vývojářská větev, slouží pro implementaci nových funkcionalit. Vývojář si ji pouští lokálně u sebe.
- **Test** – testovací větev, slouží pro opravu nalezených bugů. Je nasazena v rámci privátní sítě a zpřístupněna testerům.
- **Release** – produkční větev, kde by měla být zajištěna stabilita a požadovaná funkcionalita, slouží pro opravu hotfixů. Je nasazena v produkčním režimu a zpřístupněna široké veřejnosti.

Pro hlavní vývoj slouží Dev větev, která se po dokončení zmerguje do Test větve a následně nasadí a zpřístupní testerům. Během testování je možné na Dev větvi rozšiřovat funkcionalitu.

Během testování se nalezené bugy evidují v rámci TFS a jsou opravovány přímo na Test větvi, aby se do testovaní verze nezanášela nová funkcionalita z Dev větve a s ní případné nové chyby. Veškeré změny na Test větvi se okamžitě mergují i zpět na Dev větev.

Po dokončení testování a opravě všech nalezených chyb dojde k zmergování s Release větví a zpřístupnění široké veřejnosti. V případě, že se vyskytne nějaký akutní problém, tak se přímo na této větvi provede tzv. hotfix a úpravy se následně zmergují do obou nižších verzí.

Na obrázku 14 je názorně naznačen merge mezi větvemi.



Obrázek 14: Ukázka mergování v rámci vývojových větví.

7 Realizace

Tento projekt byl realizován na stroji s procesorem Intel Core i7 840QM s 8 GB paměti RAM a 64-bitovým OS Windows 8.1.

Pro vývoj byl zvolen programovací jazyk C# s .NET Framework 4.5 a Visual Studio 2012 Ultimate, přičemž během vývoje se přešlo na novější verzi 2013. Jako databáze byla použita SQL Server 2008 R2 Express, během vývoje se na požadavek zadavatele přešlo na vyšší verzi SQL Server 2012 Standard.

Popis jednotlivých jmenných prostorů:

`TT.Vinry.DAL.Code` – Jedná se o podpůrné třídy pro kryptování hesel a výjimky, které může tato knihovna produkovat.

`TT.Vinry.DAL.Model` – Obsahuje generovaný objektový relační model databáze a další menší modely reprezentující nezávislé entity.

`TT.Vinry.DAL.Repositories` – Obsahuje třídy, které poskytují služby dalším vrstvám aplikace (provádí se zde dotazy na databázi a vrací se konkrétní data).

`TT.Vinry.Web.Properties` – Zde je možné nastavit vlastnosti projektu jako takového (použitá verze *.NET Frameworku*, verzování, ...), dále se zde nachází `Resources` pro překlady do různých jazyků, v kterých aplikace poběží.

`TT.Vinry.Web.App_Start` – Třídy pro inicializaci různých částí aplikace (načtení scriptů a jejich minimalizace, nastavení použitých filtrů, mapování uri adres, IoC, apod.)

`TT.Vinry.Web.Content` – Umístění obrázků a CSS.

`TT.Vinry.Web.Controllers` – Controllery s akcemi, zde se řeší aplikační logika.

`TT.Vinry.Web.Extensions` – Obsahuje různé třídy, které přidávají podpůrnou funkcionalitu.

- `DecimalModelBinder` – Upravuje binding pro čísla s desetinou čárkou/tečkou u modelu, aby byla tolerována jak čárka, tak tečka.
- `HtmlHelpers` – Rozšíření Razoru.
- `ImageHelper` – Vytváření náhledů obrázků.
- `LoggingAttribute` – Logování akcí.
- `SharedAccessors` – Sdílené metody pro plnění modelů daty (používá se, pokud více controllerů používá v akcích stejné modely).

`VinryAuthorizeAttribute` – Atribut, který zajišťuje vlastní kontrolu autorizace uživatele při snaze o přístup ke controllerům/akcím.

`VinryUser` – Reprezentace uživatele systémů (uživatel je v současnosti reprezentován šifrovanou cookie).

`VinryViewEngine` – Reprezentuje předpis pro pravidla view engine.

`TT.Vinry.Web.Layout` – Obsahuje prototyp aplikace.

`TT.Vinry.Web.Models` – Obsahuje modely, které reprezentují data.

`TT.Vinry.Web.Scripts` – Umístění javascriptových souborů a knihoven.

`TT.Vinry.Web.Views` – Obsahuje jednotlivé view pro akce controllerů. Složky v tomto jmenném prostoru odpovídají názvům controllerů a jednotlivé view odpovídají jednotlivým akcím. Dále je zde složka `Share`, kde se nacházejí sdílené view, hlavně _Layout.cshtml.

`TT.Vinry.Web.Global.asax` – Inicializační třída aplikace.

7.1 Komplexnější ViewModely

Jak již bylo zmíněno výše, ViewModel třídy reprezentují data pro samostatné view. Struktura těchto tříd nemusí odpovídat tabulkám v databázi, resp. vygenerovaným třídám, které je reprezentují. Např. stránka s detailem produktu potřebuje data z tabulek PRODUCT, PRODUCTFILE, PRODUCTDATA, PRODUCTIMAGE a dalších. Ve výpisu 8 je znázorněno, jak se sestavuje ViewModel pro stránku s detailem produktu.

Nejprve se stáhne produkt. Pokud není explicitně uveden jazyk, budou stažena data ve výchozím jazyce uživatele.

```
1  try
2  {
3     var ttProduct = OwnerRepository.GetProduct(VinryUser.IdCompany, id);
4     var langs = OwnerRepository.GetLanguages(VinryUser.IdCompany);
5     var lang = idLang ?? VinryUser.DefaultLanguageId;
6     var ttProductData = ttProduct.TTProductData.FirstOrDefault(m => m.IdLanguage
          == lang);
7     var productFiles = ttProduct.TTProductFile.Where(m => m.IdLanguage == lang).
          Select(m => new ProductFileViewModel(m));
8
9     if (ttProductData == null)
10    {
11       ttProductData = new TTProductData { IdLanguage = lang };
12    }
13
14    var product = new ProductViewModel(ttProduct)
```

```
15    {
16       ProductData = new ProductDataViewModel(ttProductData),
17       ProductFiles = productFiles,
18       Languages = langs
19    };
20
21    return product;
22 }
23 catch (VinryDalException e)
24 {
25    switch (e.ExceptionType)
26    {
27       case VinryDalException.Type.ProductNotFound:
28          TempData[Message] = string.Format("Neexistujici produkt.");
29          break;
30       case VinryDalException.Type.LanguageNotExist:
31          TempData[Message] = string.Format("Neexistujici jazyk.");
32       break;
33    }
34
35    return null;
36 }
```

Výpis 8: Sestavení ViewModelu pro stránku detailu produktu.

7.1.1 Využití vygenerovaných modelů

Při vytváření ViewModelů se využívá vygenerovaných modelových tříd, jak je ukázáno ve výpisu 9. Prostřednictvím vlastností jsou zpřístupněny jen požadované hodnoty, s kterými se ve view může pracovat. Tento přístup řeší často nepřehledné předávání dat z jednoho modelu do druhého na úrovni controlleru. Ve výpisu jsou úmyslně vypuštěny atributy, aby byl čitelnější.

```
1  public class ProductViewModel : BaseViewModel
2  {
3     private TTProduct _ttProduct;
4
5     public ProductViewModel(TTProduct product)
6     {
7        _ttProduct = product;
8     }
9
10    public int Id
11    {
12       get { return _ttProduct.Id; }
13       set { _ttProduct.Id = value; }
14    }
15
16    public string ProductNumber
17    {
```

```
18      get { return _ttProduct.ProductNumber; }
19      set { _ttProduct.ProductNumber = value; }
20   }
21
22   public ProductDataViewModel ProductData { get; set; }
23
24   ...
25 }
```

Výpis 9: Sestavení ViewModelu pro stránku detailu produktu.

7.2 Chybové stránky

Je několik způsobů, jak v ASP.NET MVC může vzniknout chybová hláška, např. pro 404 platí [10]:

- Zadaná URL se nenachází v routovací tabulce,
- pro zadanou URL neexistuje příslušný controller nebo akce,
- akce vrací `HttpNotFoundResult` zavoláním metody `HttpNotFound()`,
- akce vyvolává výjimku `HttpException` s číslem 404,
- akce přepisuje vlastnost `Response.StatusCode` na 404.

Způsobů jak chybu 404 odchytit je mnoho, ale většina z nich nepokrývá všechny výše zmíněné případy. Zachytávání např. pomocí přepsání `Application_Error` metody se odmítalo vykonat při zavolání `HttpNotFound()`, protože toto není považované za chybu.

Jednoduché řešení fungující pro všechny případy je vidět ve výpisu 10. Uvedený kód se vykonává v `Global.asax`. Stejným způsobem je řešeno i odchytávání dalších HTTP stavů.

```
1  void Application_EndRequest()
2  {
3     if (Context.Response.StatusCode == (int)HttpStatusCode.NotFound)
4     {
5        ... // logování
6        Response.Clear();
7        var routeData = new RouteData();
8        routeData.Values["controller"] = "Errors";
9        routeData.Values["action"] = "NotFound";
10
11       IController errorsController = new ErrorsController();
```

```
12          errorsController.Execute(new RequestContext(new HttpContextWrapper(Context
                ), routeData));
13      }
14  }
```

Výpis 10: Vlastní odchycení HTTP chyby 404.

7.3 Routování adres

Neřeší se hostname, ale konkrétní adresa. Bez složených závorek se jedná o konstantní hodnotu, ve složených závorkách o proměnou, která může být nastavena jako nepovinná. Je nutné držet se jistých pravidel, zvláště u nepovinných položek, jinak adresa nemusí být překládána na ,,hezkou url“, ale na url s parametry nebo v horším případě nemusí vůbec fungovat.

Názorný příklad záznamu pro routování:

Pravidlo pro routování stánky s výčtem produktů:
ProductsList/{idLang}/{sortOrder}/{page}/{search}

Příklad adresy (vytvořené pomocí metody ve výpisu 11):
`http://www.vinry.eu/ProductsList/en/Weight/3/x12`

```
1  @Html.ActionLink(Controller, Action,
2     new {
3             idLang = "en",
4             sortOrder = "Weight",
5             page = 3,
6             search = "x12"
7         })
```

Výpis 11: Vytváření odkazů s využitím Razor engine.

Adresy pro přihlášení různých rolí jsou různé:

- **Login/{company}**, pro vlastníka zboží, kde company je název vlastníka zboží (subjektu).
- **LoginStore/{company}**, pro skladníka, kde company je název skladu.
- **LoginAdmin** pro přihlášení správce.

Počítá se s tím, že uživatel může měnit atributy přímo v URL, pročež se při přístupu do databáze vždy uvádí jako parametr ID uživatele, v případě skladníka i ID jeho skladu, kterým se dotaz vymezuje.

7.4 Vlastní grid

V navrženém systému je všudypřítomná práce s tabulkovým výpisem – gridem. Pro potřeby systému bylo vytvořeno rozšíření pro Razor engine, které umožňuje vygenerování gridu s podporou stránkování, řazení a vyhledávání jen na základě zadaného ViewModelu.

Ve výpisu 12 je vidět příklad pro vytvoření tabulkového výpisu všech produktů vlastníka zboží. Nejprve se určí ViewModel, v tomto případě `ProductListViewModel`, s kterým se má při generování gridu pracovat. Dále se určí akce, která se má provést při kliknutí na řazení a její parametry. Poté se volá metoda `Columns`, kde se pomocí lambda výrazů specifikují jednotlivé sloupce a sváží se s danou vlastností. Do hlavičky se použije text ze zobrazovacího atributu dané vlastnosti. U každého sloupce je možné určit několik dalších vlastností, např. že se jedná o odkaz, určí se zarovnání, formát výpisu, atd. Dále je vidět konstrukce, která dynamicky vytvoří sloupce podle záznamů v databázi – vlastníkovi zboží je vytvořeno tolik sloupců, kolik využívá v systému skladů. Tyto sloupce vypisují počty kusů produktu v konkrétních skladech.

Poté se pro tabulku určí aktuální stránka, zdroj dat, v tomto případě kolekce obsahující `ProductListViewModely` a zavolá se samotné generování.

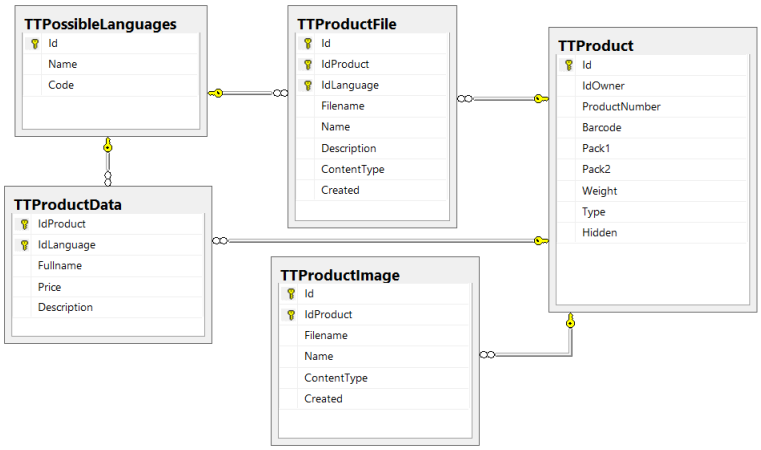
```
1  @(Html.TableFor<ProductListViewModel>("Index",new { idLang = Model.IdLang, page
       = Model.Page, search = Model.Search, sortOrder = Model.SortOrder })
2     .Columns(column =>
3     {
4        column.Expression(p => p.ProductNumber).Link(p => new { id = p.Id, @idLang
             = Model.IdLang }, "ProductDetail");
5        column.Expression(p => p.Name).Link(p => new { id = p.Id, @idLang = Model.
             IdLang }, "ProductDetail");
6        column.Expression(p => p.Pack1).AlignRight().Format("{0} ks");
7        column.Expression(p => p.Weight).AlignRight().Format("{0} kg");
8
9        foreach (var inventory in Model.Inventories)
10       {
11             var i = inventory;
12             column.Expression(p => p.Inventories.Any(m => m.Id == i.Id) ? p.
                   Inventories.FirstOrDefault(m => m.Id == i.Id).Count : 0)
13                .Title(inventory.Name).AlignRight().Format("{0} ks");
14       }
15       column.Expression(p => p.Price).AlignRight().Format("{0:C}");
16    }).Pager(Model.Page).DataSource(Model.GridProducts).ToHtml())
```

Výpis 12: Vytvoření gridu pro výpis produktů.

Výhodou tohoto řešení je univerzálnost v rámci celého projektu, variabilita a nezávislost na použití javascriptu a AJAX dotazů.

7.5 Jazykové mutace ukládaných dat

Pro ukládání informací o produktu byla vytvořena struktura, viz obrázek 15, kde se řeší i ukládání jazykových mutací. V praxi dochází k tomu, že při dotazování na produkt je vždy specifikován jazyk, který se má použít pro připojení jazykově závislých dat.



Obrázek 15: Struktura pro ukládání jazykových mutací.

7.6 Zabezpečení (reakce na TOP 10 OWASP)

Při sběru požadavků na podobu systému byla opakovaně zdůrazňována bezpečnost. Důvodem je kromě jiného práce s daty, jejichž neoprávněná modifikace by mohla způsobit finanční ztrátu.

Kvalitu zabezpečení webového systému jde jen velmi obtížně změřit, proto se v rámci této práce vycházelo z doporučení mezinárodní neziskové organizace OWASP (Open Web Application Security Project)[12], která se zabývá zlepšováním bezpečnosti softwaru. V rámci této organizace vznikl dokument Top 10 popisující deset nejkritičtějších problémů dnešních webových aplikací. Tento dokument posloužil jako standard pro zajištění požadované funkcionality.

[12] `https://www.owasp.org/index.php/Main_Page`

Konkrétní řešení jednotlivých problémů:

A1-Injection

Jako základní ochrana proti napadnutí formou SQL injection slouží výhradní užívání Entity Frameworku pro přístup k datům, který sám o sobě zajišťuje velmi pokročilou prevenci takovýchto útoků.

Při psaní vlastních SQL dotazů se vkládaná data nezadávají přímo do dotazu, ale prostřednictvím parametrů (`SqlParameter`). Tím je zabezpečeno, že nejsou dotazy žádným způsobem upraveny a vykonají pouze požadovanou akci.

A2-Broken Authentication and Session Management

Pro přihlášení uživatelů se využívá metoda Forms Authentication s využitím autentizační cookie.

Zprostředkování autentizace a autorizace se řeší prostřednictvím ASP.NET Membership providera a Role providera. Tyto třídy jsou upraveny pro použití s vlastní databází.

Hesla se ukládají jako PBKDF2 hashe, které se před kryptováním solí.

A3-Cross-Site Scripting (XSS)

ASP.NET MVC v základu zakazuje posílání potenciálně nebezpečného kódu ze strany uživatele (HTML značky, scripty, SQL dotazy atd.). Tato ochrana se dá explicitně vypnout pro zvolené formulářové prvky. U žádného uživatele toto nikdy nenastává.

A4-Insecure Direct Object References

Všechny akce se vždy vážou na konkrétního přihlášeného uživatele, který se rozlišuje pomocí autentizační cookie. Dále se u každé akce řeší autorizace pomocí autorizačního atributu, který definuje, jaké role mohou danou akci vykonat. V případě nesplnění požadavků je uživatel přesměrován na přihlašovací stránku.

A5-Security Misconfiguration

Všechny vývojové větve využívají stejný proces nasazení aplikace. Existuje pouze jeden základní konfigurační soubor, který se pro různé větve modifikuje. Změny konfiguračního souboru jsou pro různé větve minimální. Verze jednotlivých součástí se evidují jako nuget balíčky, čímž je zajištěno, že při přechodu na novější verze bude projekt aktualizován i na dalších vývojových větvích.

A6-Sensitive Data Exposure

Uživateli se posílá minimum citlivých dat, které slouží pro jeho autentizaci v rámci aplikace. Tato data jsou šifrovaná. Pro komunikaci s produkční verzí systému se zakoupí certifikát podepsaný důvěryhodnou certifikační autoritou a pro přístup k systému se bude využívat protokolu TLS.

A7-Missing Function Level Access Control

Všechny controllery, které přistupují k citlivým datům jsou opatřeny autorizačním atributem.

A8-Cross-Site Request Forgery (CSRF)

Každý formulář je opatřen unikátním autorizačním tokenem (`AntiForgeryToken`), který se při jeho zpracovávání validuje. To samé se děje při volání akcí prostřednictvím AJAX. [22]

A9-Using Components with Known Vulnerabilities

Aplikace využívá nejnovějších technologií, které byly v době jejího vývoje k dispozici. Většina základní funkcionality je řešena od základu podle zavedených praktik a doporučení od zaměstnavatele.

A10-Unvalidated Redirects and Forwards

V případě, že je aplikaci zadáno URL jako parametr (např. při přihlašování), ověří si prvně sytém, zda se jedná o lokální adresu pomocí metody `IsLocalUrl`. Pokud se jedná o lokální adresu, dojde k přesměrování, v opačném případě je uživatel přesměrován na úvodní stránku.

7.7 Použité knihovny

Při vývoji se využilo několika externích knihoven a nástrojů, konkrétně se jedná o:

- **ServiceStack JsonSerializer**[13]
  Rozšíření pro snazší a lépe optimalizovanou serializaci dat do JSON. Využívá se kromě práce s AJAX dotazy i pro ukládání dodatečných dat v autorizační cookie.
- **Simple Injector**[14]
  Používá se pro zpřístupnění Repository rozhraní z projektu `TT.Vinry.DAL`. Slouží k vytvoření `DbContextu` při každém zavolání stránky (`PerWebRequest`) a zprostředkovává tak bezpečný přístup k databázi.
- **PagedList**[15]
  Využívá se u implementace vlastního gridu. Značně usnadňuje tvorbu stránkování.
- **MVC TreeView Helper**[16]
  Rozšíření pro snadné vytváření stromových struktur. Využívá se u práce s tagy.

[13] `http://mono.servicestack.net/docs/text-serializers/json-serializer`
[14] `https://simpleinjector.codeplex.com`
[15] `http://github.com/TroyGoode/PagedList`
[16] `http://mvctreeview.codeplex.com`

8 Testování

Součástí návrhu bylo vytvoření několika ukázkových testovacích scénářů, které slouží jako podklad pro interní uživatelské testování.

Každý scénář obsahuje na začátku metodické pokyny, aby byl jasný způsob, jak s ním nakládat. Pod každým pokynem se nachází místo pro poznámku, kterou je vhodné při zadání „Ne“ vyplnit. V tabulce 9 je příklad jednoduchého testovacího scénáře.

8.1 Testování v beta verzi

Jakmile byla implementována veškerá definovaná funkcionalita, byl produkt nasazen u zadavatele s využitím VPN (Virtual Private Network) pro omezení přístupu z vnějšku. Pro testování byl vygenerován prostřednictvím CAcert[17] testovací certifikát, který je vhodné přidat mezi důvěryhodné certifikáty u všech počítačů, které k aplikaci přistupují. Pokud není certifikát přidán, prohlížeč má snahu uživatele na stránku nepustit. Do ostrého provozu se počítá s nakoupením certifikátu podepsaného uznávanou certifikační autoritou (CA).

Zadavatel měl možnost si systém vyzkoušet a vyjádřit se k němu.

8.1.1 Konfigurace počítače

Testování aplikace proběhlo na počítači s následující konfigurací. Konfigurace se liší od té uvedené ve specifikaci požadavků, jelikož v průběhu vývoje byl testovací stroj aktualizován:

- **Procesor:** Intel Xeon CPU 3,0GHz
- **Pamět RAM:** 9 GB
- **Operační systém:** Windows Server 2012 Standard
- **Webový server:** Internet Information Services (IIS) 8.0
- **Databáze:** SQL Server 2012 Standard
- **Pevný disk:** 500 GB

[17] `http://www.cacert.org`

Test změna nastavení (vlastník zboží)	
Kliknutím na záložku „Nastavení“ v horní části obrazovky se zobrazí stránka s osobními údaji o přihlášeném uživateli.	Ano / Ne
Na kartě „Editovat můj profil“ lze upravovat nastavení uživatele.	Ano / Ne
Editovat můj profil – zvolte volbu ,,změnit heslo“. Nabídl vám systém zadání stávajícího a nového hesla?	Ano / Ne
Po změně hesla klikněte na tlačítko ,,Odhlásit“ v horní části obrazovky. Zobrazí se stránka pro přihlášení, zadejte nové přihlašovací údaje. Podařilo se vám přihlásit se?	Ano / Ne
Editovat můj profil – je možné změnit výchozí nastavení jazyka a změna se ihned projevuje.	Ano / Ne
Editovat můj profil – je možné změnit počet záznamů na stránku a změna se v tabulkách projevuje.	Ano / Ne
Editovat můj profil – pole označená hvězdičkou jsou povinná. Bez nich nelze formulář odeslat.	Ano / Ne
Na kartě „Můj subjekt“ lze upravovat nastavení celého subjektu.	Ano / Ne
Můj subjekt – lze přidávat/odebírat uživatele subjektu (kromě vašeho účtu).	Ano / Ne
Můj subjekt – při vytváření nového účtu s již existujícím uživatelským jménem jste byl upozorněn na nutnost zadat unikátní jméno.	Ano / Ne
Můj subjekt – je možné změnit číslo účtu.	Ano / Ne
Můj subjekt – je možné změnit přednastavenou hodnotu DPH.	Ano / Ne
Můj subjekt – pole označená hvězdičkou jsou povinná. Bez nich nelze formulář odeslat.	Ano / Ne
Zkuste odeslat formulář bez vyplnění povinných údajů. Formulář se nepovedlo odeslat a povinné údaje byly zvýrazněny – je tomu tak?	Ano / Ne

Tabulka 9: Příklad jednoduchého testovacího scénáře.

8.2 Výsledky testování

Při vývoji se simulovalo značné zaplnění databáze cyklickým nahráváním produktů, řádově sta tisíce produktů, aby se ověřila její rychlost při tabulkovém výpisu, řazení a vyhledávání. Zpomalení nebylo pouhým okem patrné, jednalo se řádově o zlomek sekundy.

8.2.1 Zpětná vazba

Pro evidování zpětné vazby posloužil soubor formátu Excel, kde se zaznamenávaly veškeré připomínky. Tento soubor zadavatel sdílel prostřednictvím online služby Office 365.

Pro větší přehlednost a svázání s projektem se úkoly a bugy průběžně přepisovaly do TFS, jak bylo navrženo v sekci 6.10. Výhodou takového řešení byla i možnost svázání veškerých zasílaných úprav s konkrétními úkoly/bugy. Ukázka zaznamenané zpětné vazby v TFS je vidět na obrázku 16.

Assigned to me

ID	Work Item Type	Title	Assigned To	State	Area Path
16	Task	Přidat k vlastníkovi zboží a skladníkovi do profilu email.	Ondřej Trhoň	New	Virtual Inventory
17	Task	Grafické reporty.	Ondřej Trhoň	New	Virtual Inventory
18	Task	Exporty reportů do pdf, csv a excelu.	Ondřej Trhoň	New	Virtual Inventory
19	Bug	Při vytvoření zákazníka 404 chyba, viz obr.	Ondřej Trhoň	Active	Virtual Inventory
20	Task	Ruční korekce počtu kusů zboží.	Ondřej Trhoň	New	Virtual Inventory
21	Task	Přidat komentáře u objednávek a dodávek.	Ondřej Trhoň	New	Virtual Inventory

Obrázek 16: Výpis úkolů a bugů po nasazení do testovacího provozu (TFS).

9 Závěr

Cílem této práce bylo vytvořit webový systém pro správu virtuálních skladů určený zákazníkům v rámci Evropské Unie, který by poskytoval jednotné prostředí pro práci se sklady v různých zemích a jazycích.

Samotné realizaci předcházelo seznámení se s řízením dodavatelského řetězce a e-logistikou, konkrétně s outsourcingem skladových prostor.

Následovalo představení aktuálních trendů v oblasti tvorby webových aplikací a porovnání jednotlivých technologií. Dále došlo k seznámení se s firmou zadavatele a definování konkrétní funkcionality požadovaného systému od vize až po specifikaci konkrétních požadavků. Pro potřeby zadavatele bylo provedeno základní porovnání možností nasazení konečného produktu.

Po vytvoření specifikace požadavků byly analyzovány konkrétní možnosti vývoje včetně vzhledu s ohledem na zvolenou technologii ASP.NET MVC. Zároveň bylo vytvořeno několik uživatelských testovacích scénářů. Na základě předchozích poznatků se realizovala aplikace, jejíž funkcionalita se průběžně prezentovala zadavateli. Pro správu verzí zdrojových kódů se využilo nástroje TFS.

Po dokončení práce byl výsledný systém zprovozněn v testovacím režimu, aby si ho mohl zadavatel osobně vyzkoušet a poskytnout zpětnou vazbu. Do TFS bylo zaevidováno několik postřehů pro další rozvoj aplikace. Součástí práce není samotný překlad do různých jazyků, je pouze poskytnuta jednoduchá struktura, kde je možné překlady zadat.

Práce ve výsledku splňuje požadavky, které jsou uvedeny na jejím začátku, přestože její tvorba trvala déle, než se původně předpokládalo.

Použitá literatura

[1] KOCH, Richard. *The 80/20 principle: the secret of achieving more with less*. Reprint. London: N. Brealey, 1998. ISBN 18-578-8167-2.

[2] LAMBERT, Douglas. *Logistika: [příkladové studie, řízení zásob, přeprava a skladování, balení zboží]*. 1. vyd. Praha: Computer Press, 2000, 589 s. ISBN 80-722-6221-1.

[3] KYPSOŇ, Radek. *Metodika implementace e-logistiky ve vertikálních distribučních řetězcích se zaměřením na firmy menší a střední velikosti*. 1. vyd. Brno: VUTIUM, 2002. 33 s. ISBN 80-214-2143-6. Zkrácená verze PhD Thesis. Vysoké učení technické v Brně.

[6] BASL, Josef a Roman BLAŽÍČEK. *Podnikové informační systémy: podnik v informační společnosti. 3.*, aktualiz. a dopl. vyd. Praha: Grada, 2012, 323 s. ISBN 978-80-247-4307-3.

[10] FREEMAN, Adam. *Pro ASP.NET MVC 4*. 4. vyd. Berkeley, Calif: Apress, 2012. ISBN 978-143-0242-369.

[12] TIWARI, Shashank C. Professional nosql. 1st ed. Indianapolis, IN: Wiley Pubishing, Inc., 2011, p. cm. ISBN 04-709-4224-X.

[18] AGEL, Christian, Morgan SKINNER, Karli WATSON, Jay GLYNN a Bill EVJEN. *C# 2008: programujeme profesionálně*. Vyd. 2. Brno: Computer Press, 2013, 1904 s. ISBN 978-80-251-2401-7.

[20] KALUŽA, Jindřich a Ludmila KALUŽOVÁ. *Modelování dat v informačních systémech*. 1. vyd. Praha: Ekopress, 2012, 125 s. ISBN 978-80-86929-81-1.

[21] CONOLLY, Thomas, Carolyn E BEGG a Richard HOLOWCZAK. *Mistrovství – databáze: profesionální průvodce tvorbou efektivních databází*. Vyd. 1. Brno: Computer Press, 2009, 584 s. ISBN 978-80-251-2328-7.

Ostatní zdroje

[4] BUXBAUM, P. A. Digital Logistics –Value Creation in the Freight Transport Industry, *Eyefortransport Conference*, Las Vegas, Květen, 2001.

[5] KRIŽKO, Ivo. SCM: Supply Chain Management. In: *SystemOnLine* [online]. 2002 [cit. 2014-04-11]. Dostupné z: `http://www.systemonline.cz/clanky/scm-supply-chain-management.htm`

[7] ZAHÁLKA, Luděk a Rostislav SCHWOB. Warehouse management. *SystemOnLine: Ekonomické a informační systémy v praxi* [online]. 9/2009 [cit. 2014-03-13]. Dostupné z: http://www.systemonline.cz/erp/warehouse-management.htm

[8] Usage of server-side programming languages for websites. *W3Techs* [online]. 2014 [cit. 2014-04-11]. Dostupné z: http://w3techs.com/technologies/overview/programming_language/all

[9] ESPOSITO, Dino. Exploring ASP.NET 4.0—Web Forms and Beyond. [online]. [cit. 2014-04-21]. Dostupné z: http://msdn.microsoft.com/en-us/magazine/ee431529.aspx

[11] Market Share. *MySQL* [online]. 2008 [cit. 2014-04-28]. Dostupné z: http://www.mysql.com/why-mysql/marketshare

[13] KUCINSKAS, Darius. Entity Framework 6 vs NHibernate 4. In: *Devbridge Group* [online]. 2014 [cit. 2014-04-18]. Dostupné z: http://www.devbridge.com/articles/entity-framework-6-vs-nhibernate-4

[14] ADO.NET and LINQ to SQL. *Microsoft Developer Network* [online]. 2013 [cit. 2014-03-06]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/bb386944(v=vs.110).aspx

[15] JAVOREK, Jan. CSS preprocesory: méně psaní, vyšší efektivita. In: *Zdroják.cz* [online]. 2011 [cit. 2014-04-21]. Dostupné z: http://www.zdrojak.cz/clanky/css-preprocesory-mene-psani-vyssi-efektivita

[16] MICHÁLEK, Martin. K čemu je dobrý Bootstrap a frontend frameworky?. In: *Zdroják.cz* [online]. 2013 [cit. 2014-04-21]. Dostupné z: http://www.zdrojak.cz/clanky/k-cemu-je-dobry-bootstrap-frontend-frameworky

[17] ANDERSON, Rick. Bundling and Minification. In: *ASP.NET* [online]. 2012 [cit. 2014-04-26]. Dostupné z: http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification

[19] SHEKHAWAT, Sandeep Singh. CRUD using the Repository Pattern in MVC. In: *C# Corner* [online]. 2013 [cit. 2014-04-28]. Dostupné z: http://www.c-sharpcorner.com/UploadFile/3d39b4/crud-using-the-repository-pattern-in-mvc

[22] WASSON, Mike. Preventing Cross-Site Request Forgery (CSRF) Attacks. In: *ASP.NET* [online]. 2012 [cit. 2014-05-01]. Dostupné z: http://www.asp.net/web-api/overview/security/preventing-cross-site-request-forgery-(csrf)-attacks

Seznam zkratek

Akronymy použité v tomto dokumentu:

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
CA	Certification Authority
CRUD	Create, read, update and delete
CSS	Cascading Style Sheets
DAL	Data Access Layer
DBMS	Database Management System
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
IIS	Internet Information Services
IoC/DI	Inversion of Control/Dependency Injection
Java EE	Java Platform, Enterprise Edition
JSP	JavaServer Pages
MIT	Massachusetts Institute of Technology
MVC	Model-Ciew-Controller
NoSQL	Not Only SQL
ORM	Object-Relational Mapping
OWASP	Open Web Application Security Project
RDBMS	Relational Database Management System
REST	Representational State Transfer
RIA	Rich Internet Application
SCM	Supply Chain Management

SCOR Supply Chain Operation Model

SOAP Simple Object Access Protocol

SQL Structured Query Language

SVN Apache Subversion

TFS Team Foundation Server

TMS Transportation Management System

VPN Virtual Private Network

VPS Virtual Private Server

WCF Windows Communication Foundation

WF Windows Workflow Foundation

WMS Warehouse Management System

WPF Windows Presentation Foundation

WSDL Web Services Description Language

A Stručná uživatelská příručka

Aplikace „Virtuální sklady“

Pro běh aplikace je nutné nejprve na SQL Serveru vytvořit příslušnou databázi. K tomu slouží přiložený SQL dump script, viz D, složka *Aplikace*.

Poté se musí upravit v souboru web.config informace o použité databázi – ta se eviduje jako záznam nazvaný connectionString.

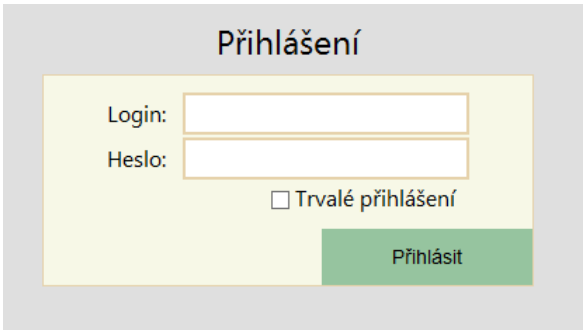
Pro běh webové aplikace je potřeba IIS Express 7 nebo novější. Případné nastavení certifikátu se řeší na úrovni aplikačního serveru IIS.

Po spuštění přejděte na stránku `http://hostname:port/LoginAdmin`, měla by se vám objevit přihlašovací obrazovka, viz obrázek 17.

Výchozí přihlašovací údaje administrátora systému jsou:

- **Login:** spravce
- **Heslo:** Spravce

Tyto údaje je možné editovat v souboru *web.config*.



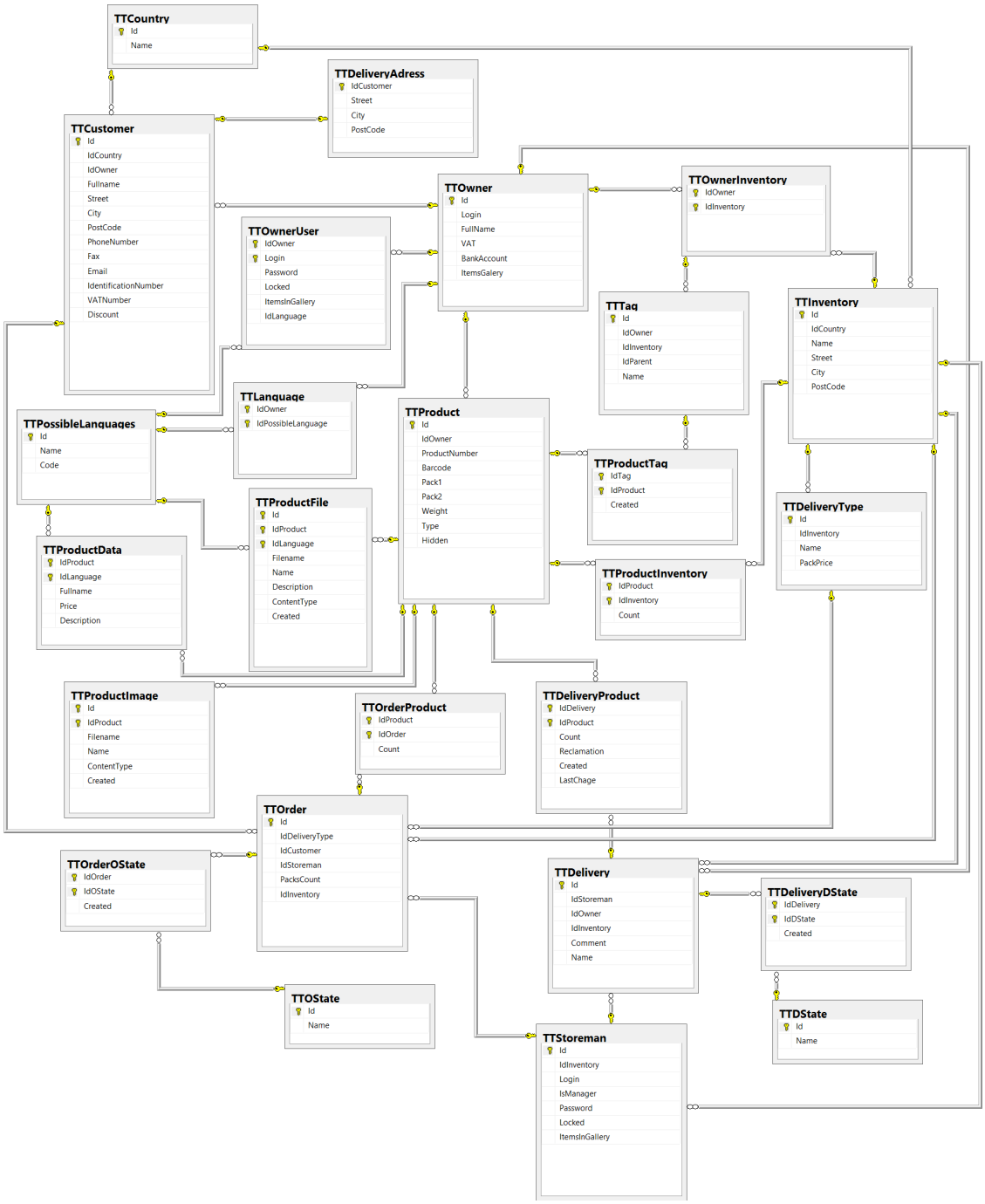
Obrázek 17: Přihlašovací stránka.

Po přihlášení je možné editovat číselníky v databázi a vytvořit tak příslušné jazyky, sklady, skladníky a vlastníky zboží.

Adresa pro přihlášení skladníků je `http://hostname:port/LoginStore/{company}` a vlastníků zboží `http://hostname:port/Login/{company}`.

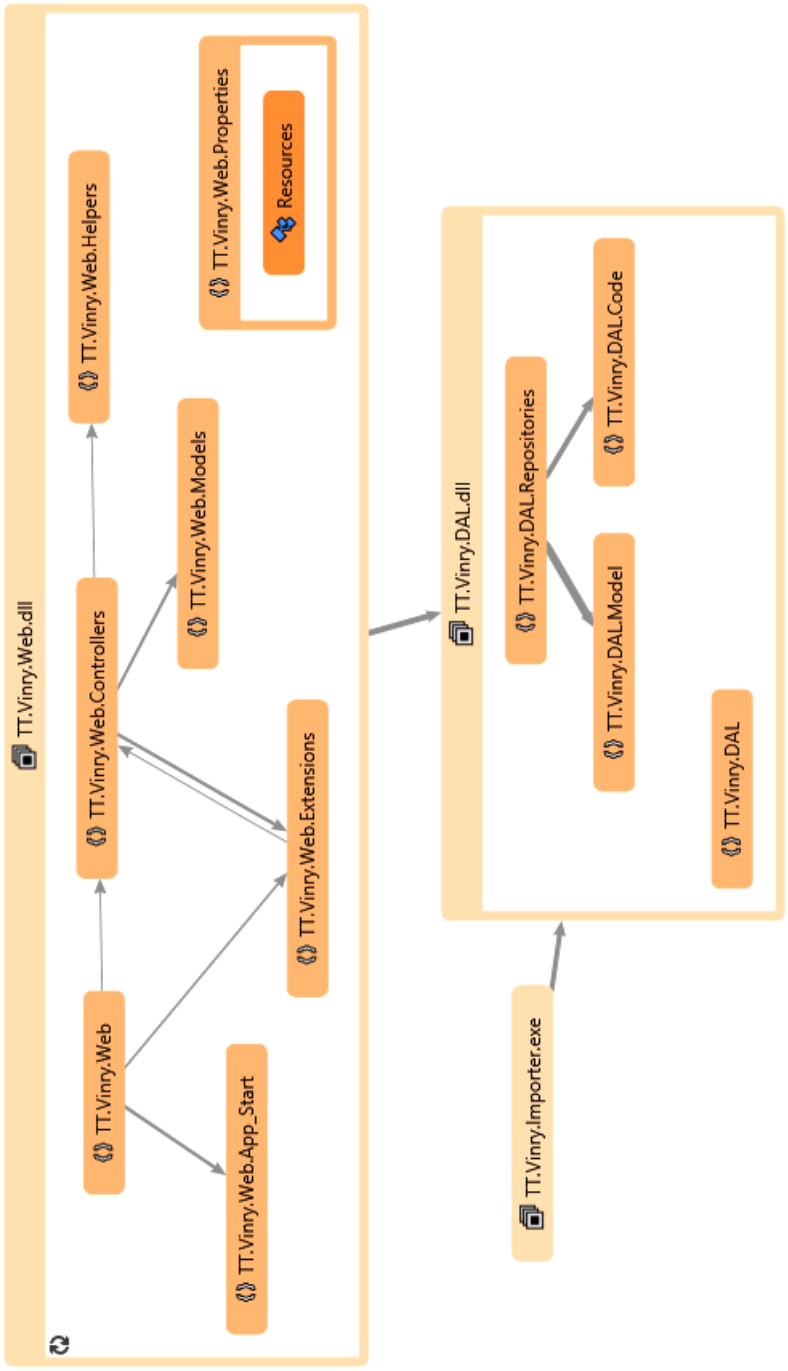
Ovládání je jednoduché a intuitivní – možnosti jednotlivých uživatelů jsou popsány v sekcích 6.3, 6.4 a 6.5.

B ERA model databáze



Obrázek 18: Navržená struktura databáze.

C Graf závislostí základních jmenných prostorů



Obrázek 19: Graf závislostí jmenných prostorů.

D Obsah CD

Přílohou tohoto dokumentu je i CD, které má následující strukturu:

- **Aplikace**
  - **TT.Vinry**
    - Obsahuje projekt (solution) ve Visual Studiu 2013.
    - Součástí solution jsou všechny tři části zmíněné v sekci 6.6.
    - Pro svůj běh vyžaduje IIS Express 7 nebo novější.
    - Je nutné upravit web.config pro připojení k databázi.
  - Obsahuje script.sql pro vytvoření schéma databáze.
- **Dokumenty**
  - obsahuje dokument dokument specifikace požadavků *Specifikace pozadavku.docx.*
  - obsahuje text diplomové práce *DIP.pdf.*
  - **Zdroj**
    - Obsahuje zdrojové kódy pro diplomovou práci ve formátu Lyx 2.0.6.
    - Obsahuje obrázky použité v tomto dokumentu.