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

Framework pro deklarativní vývoj aplikací pomocí XML

Diplomová práce

Autor: Jan, Stohanzl
Studijní obor: Aplikovaná informatika (ai2-k)

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Odborný konzultant:

Hradec Králové duben 2016

Prohlášení:

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

V Hradci Králové dne 22. 04. 2016 Jan Stohanzl

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Pavlu Kříži, Ph.D. za metodické vedení práce a neocenitelnou pomoc při sestavování výsledného textu.

Anotace

Framework pro deklarativní vývoj aplikací pomocí XML slouží pro jednoduché vytvoření webové Single Page Application, která má napodobovat zdání desktopového prostředí. Uživateli by měl nabídnout možnost pracovat na tenkém klientovi téměř jako na klientech silných. Framework má být určen jen pro úzce specializovanou skupinu uživatelů se znalostmi SQL a HTML (popř. JavaScript) a snaží se odstínit návrháře výsledné aplikace od vlastního programování. Formou konfiguračních XML by měl být uživatel schopen vytvořit i celkem složité aplikace bez znalostí programovacího jazyka či podpory programátora. Pro firmy to přináší možnost rozšířit řady vývojářů o své odborníky na SQL databáze.

Aplikace jako taková by měla být řízena pomocí deklarovaných událostí, které se budou lišit podle nastavených práv a pravidel.

Annotation

Title: XML-based Framework for Declarative Application Development

The final product of this work, i.e., Framework for Declarative Application Development by XML, is intended to simply create a Single Page Application, which is to mimic the appearance of a desktop environment. It should offer the user the opportunity to use thin clients in almost the same way as thick clients. The framework should be used only by a highly specialized group of users with a wide knowledge of SQL and HTML (or JavaScript). It attempts to separate the final application designers from the programming itself. By using configuration XML files, the user should be able to create relatively complex applications without knowledge of the programming language or without support from the programmer. This allows businesses to increase the number of developers by employing their own SQL database experts. The application itself should be able to be controlled by using declarative events, which will differ depending on the rights and rules.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V mnoha společnostech, zabývajících se tvorbou softwaru, dochází k situaci, kdy dlouhodobě vyvíjenou desktopovou aplikaci je nutné přizpůsobit moderním trendům a vytvořit pro ni webové grafické uživatelské rozhraní (GUI). Ať je již důvod tvorby webové aplikace konkurenční boj, nebo požadavky zákazníka, vývojáře stojí před nelehkým rozhodnutím, kdy je nutné zvážit, jakým směrem obrátit vývoj. Situaci lze řešit několika způsoby:

- Vývoj aplikace lze rozštěpit na verzi desktopovou a verzi webovou
- Pomocí dostupných nástrojů doplnit aplikaci o webové GUI
- Pokud to použitá technologie dovoluje, lze aplikaci přizpůsobit již existujícími nástroji a vytvořit více uživatelských rozhraní

Při rozštěpení vývoje aplikace na vývoj webové a desktopové aplikace vznikají dvě nezávislé aplikace, vyvíjené dvěma týmy, čímže se zvyšuje riziko chybné implementace požadavků, vyplývající například z chybné dokumentace či implementace, a při specifické kombinaci dat se může projevit odlišné chování. Koordinovat dva týmy vývojářů je složitá záležitost, vliv hraje i organizační struktura společnosti.

Vytvoření webového rozhraní nad existující aplikací je jedna z možností, kdy se předpokládá, že existující aplikace má stabilní datový model a známou business logiku. Výsledkem takového rozhodnutí bude sestavení grafického uživatelského rozhraní pro webové aplikace nad datovým modelem desktopové aplikace a dvě víceméně nezávislé aplikace. Nevýhodou vybraného řešení je nutnost koordinovat změny v obou aplikacích, změny nad datovým modelem jedné aplikace musí být synchronizovaně promítnuty i do druhé aplikace, pokud mají pracovat společně.

Poslední variantou je použití technologie, která dovoluje vyvíjení aplikací pro více platforem najednou. Z hlediska údržby aplikací nejlepší volbou. I tato varianta však má své nevýhody. Pokud je aplikace vyvíjena pro více platforem

najednou, musí být zohledněn fakt, že pokud mají být vývojářské technologie funkční na více platformách stejně, musí být omezeny vývojářské prostředky podle nejomezenější, podporované platformy. Dalšími problémy jsou obtíže s adaptací a kompatibilitou jednotlivých verzí vývojářských prostředků, či nedostatek kvalifikované pracovní síly. Zatímco naučit se základy SQL, pochopit datové modely aplikací není až tak složité, naučit se chápat principy programování a držet krok s moderními trendy je složitější a nákladnější.

Velmi často se stává, že zákazníci neprovádějí pravidelně předepsané aktualizace. Pokud se rozhodnou rozšířit stávající desktopové aplikace o aplikace webové, vznikají situace, kdy stávající zákaznické systémy nejsou kompatibilní s verzí webové aplikace vyvíjené nad aktualizovaným datovým modelem. Následně se řeší situace, kdy společnost vyvíjející webovou aplikaci bude muset přizpůsobit webové rozhraní vůči stávajícímu zákaznickému systému. To vede k rozštěpení verzí. U moderní aplikační architektury, kde datové modely bývají pevně sepjaty s objektovými modely aplikace, je nutné provést opravu, nový překlad aplikace – a nové nasazení. Pokud by byl deklarativní Framework správně navržen, bylo by modifikace možné provést přímo na místě u zákazníka, bez nutné přítomnosti programátora.

Např. při vývoji aplikace pomocí nástroje DevExpress je kladen důraz na objektový model. Před spuštěním vlastního programu dojde ke kontrole objektového modelu aplikace vůči datovému modelu v cílovém datovému úložišti. V případě, že je nalezen byť minimální rozdíl, aplikaci se nepodaří spustit. Rozdílem chápána i rozdílná velikost definovaného sloupce v tabulce. V takovém případě je nutné vytvořit novou větev stávající aplikace podle instalované verze a doplnit funkcionalitu verze požadované. Celý proces vede k vytvoření rozdílové analýzy, zapracování požadovaných změn a následná publikace.

Framework pro deklarativní vývoj aplikací pomocí XML má sloužit právě pro situace, kdy nelze původní aplikaci modifikovat nebo převést. Snaží se těžit z faktu, že v každé podobné společnosti, se nachází dostatek vzdělaných specialistů, kteří chápou datové modely vyvíjených aplikací, orientují se

v aplikačních procesech a jsou schopni pochopit principy konfiguračních XML souborů. Specialista pak pomocí konfiguračních souborů bude schopen vytvořit na základě datového modelu existující aplikace, konfiguraci webového GUI, která se postará o správnou interpretaci požadovaných úkonů. Není vyloučena ani možnost používání frameworku jako samostatného nástroje na tvorbu webové aplikace.

Tato práce se bude zabývat otázkami vedoucí k vytvoření popsaného nástroje pro tvorbu webových aplikací. Bude proveden technologický průzkum pro vyhledání existujících nástroj a technologií, ze kterých by bylo možné čerpat vědomosti při tvorbě vlastního nástroje sloužícího tomuto účelu. Bude provedena analýza předpokládaných požadavků na výsledný produkt – webové aplikace řízené konfiguračními strukturovanými soubory.

Smyslem snažení je vytvoření aplikace, sloužící k jednoduchému vytvoření webové aplikace bez nutných znalostí programovacích jazyků, která by při práci navozovala pocit práce na desktopové aplikace. Cílem není vytvoření plnohodnotné náhrady desktopové aplikace, jen v rámci možností vyšší dostupnosti stávající desktopové aplikace, nebo zpřístupnění široké veřejnosti přes internet.

2 Existující nástroje

V této kapitole budou představeny technologie, které již existují, popř. technologie, se kterými se při tvorbě webových aplikací lze setkat. Vyjmenovat všechny by nebylo možné, proto se práce bude zabývat prostředky, které byly dostupné a dostatečně známé v době vzniku textu.

2.1 MDA (Model Driven Architecture)

Mezi existující nástroje, které by umožnily nad existujícím modelem vytvořit aplikaci, se dají zařadit aplikace vycházející z architektury. U této architektury se předpokládá abstrakce modelu zachycená pomocí UML a následné generování kódu aplikace. Jedním z těchto nástrojů je Enterprise Architect, který umožňuje pomocí reverzního inženýrství (Reverse Engineering) analyzovat databázový model. Analýzou databázového modelu lze celkem přesně zachytit vztahy mezi jednotlivými tabulkami, co však z modelu nelze vyčíst, je aplikační logika. Ta se musí dodatečně do modelu doplnit, což, díky snaze o nezávislost na cílové platformě, je téměř nadlidský úkon. Pokud se ji však zachytit povede, lze vygenerovat kód pro požadovanou platformu [1].

2.2 Databázové editory

Databázové editory slouží převážně k administraci databází, umožňují ale celkem přehledně procházet datový model databáze a modifikovat záznamy. Vznikají oficiální i neoficiální nástroje umožňující náhled na data, modifikaci tabulek i databáze samotné. Samotné editory jsou jen prostým nástrojem postrádajícím vztah k obsahu tabulek. Dokážou přihlášenému zkušenému uživateli zpřístupnit informace, které by jinak musel získávat pomocí konzole. Ve chvíli, kdy však editační nástroj začne spolupracovat s databázovým schématem a umožní instalovat rozšíření upravující pohled na data, stává se z něho nástroj, který už

nemusí sloužit jen návrhářům, ale i koncovým uživatelům. Mezi takové nástroje lze zařadit Adminer[1].

Sám o sobě je Adminer vylepšený phpMyAdmin, ovšem díky možnosti rozšíření umožňuje modifikovat pohledy na data a svým způsobem do tohoto rozhraní vtěsnat aplikační logiku.

2.3 Používané vývojové nástroje

Pod vývojovými nástroji si lze představit téměř vše, co lze použít k usnadnění tvorby aplikace. Takovým nástrojem může být ve finále i textový editor. Textový editor nám pomůže při tvorbě či editaci zdrojových kódů, ale samostatně fungující program z něj učiní až kompilátor programovacího jazyka. Dle společnosti TIOBE je v současné době nejžádanější programovací jazyk JAVA [2] a jeho popularita má stále stoupající tendenci. Svůj podíl na stoupající popularitě může mít i skutečnost, že aplikace pro operační systém Android jsou vyvíjeny právě v jazyce JAVA.

Tabulka 1 – 10 nejpoužívanějších programovacích jazyků, zdroj TIOBE

Pořadí	Programovací jazyk	Hodnocení
1	JAVA	20,528 %
2	C	14,600 %
3	C++	6,721 %
4	C#	4,271 %
5	Python	4,257 %
6	PHP	2,768 %
7	Visual Basic .NET	2,561 %
8	JavaScript	2,333 %
9	Perl	2,251 %
10	Ruby	2,238 %

[1] Oficiální webová stránka nástroje Adminer https://www.adminer.org/

2.4 Využívané technologie a přístupy

Tvorba webových aplikací je v poslední době směřována na vývoj webových aplikací určených pro běžné prohlížení na PC a na vývoj aplikací určených především pro mobilní platformu. Aplikace určené pro mobilní platformu jsou limitovány především velikostí obrazovky a výkonem zařízení. Ačkoliv jsou dnes v prodeji mobilní zařízení, u kterých je rozlišení obrazovky schopno zobrazit „Full HD“ obraz (1920 x 1080), tak je třeba si uvědomit i jemnost takového displeje. Lidské oko je schopno vnímat rozdíly přibližně do 300 PPI (Pixel Per Inch) [3]. U rozlišení Full HD na 5,5 palců se dostáváme na hranici asi 400 PPI, což je o 100 PPI více, než je hranice vnímání lidského oka. Z toho vyplývá nutnost použití většího písma, aby bylo možné text přečíst, čímž se zmenšuje zobrazitelná plocha.

V dnešní době se vyskytuje alespoň jeden nástroj na tvorbu aplikací, který je postaven na principu skenování datových tříd a na základě toho vygeneruje potřebné datové úložiště a ovládací prvky – eXpressApp Framework [4].

Aplikace lze vyvíjet různými způsoby. Nejlepším způsobem pro udržení správného programování je využívání dostupných nástrojů – frameworků, integrující „*best practice*“ postupy. Nelze jednoznačně stanovit, jaké nástroje jsou nejlepší, lze však souhrnně popsat vlastnosti těch nejrozšířenějších.

2.4.1 MVC (Model View Controller)

MVC je v dnešní době snad nejčastěji používaná architektura, nejen webových, aplikací. Návrhové vzory či nástroje pro tvorbu MVC aplikací lze nalézt pro téměř všechny dostupné programovací jazyky, jako jsou JAVA, PHP, C#, …

MVC není nová technologie, jedná se o technologii prvně představenou již v 70. letech 20. století.

- Model – shrnuje všechna data či datové struktury, se kterými aplikace pracuje. Nezáleží na tom, kde jsou data fyzicky uložena, pouze nás zajímá

jejich struktura, která má přímou návaznost na ostatní části. Modelem jsou chápány nevizuální třídy.

- View – ve chvíli, kdy je k dispozici datová strukturu aplikace, je potřeba data správným způsobem prezentovat. K tomuto účelu slouží View, čili pohledy, které nám říkají, jakým způsobem se mají data zobrazit. View ve své podstatě je HTML stránka/formulář, představující uživatelsky viditelnou část programu.
- Controller – aby bylo možné data aktualizovat, je nutné řídit akce vedoucí k udržení konzistence dat. Controller se stará o spouštění a udržování dat. Controller je nevizuální komponenta, nicméně její přítomnost u webových aplikací je nepřehlédnutelná (je to např. součást URL).

Velmi výhodná je přenositelnost kódu mezi aplikacemi a kód již jednou napsaný se dá využít jinde, což urychluje práci při tvorbě nových aplikací.

Mezi typické zástupce MVC patří Spring Framework MVC pro Javu, ASP. NET MVC, Castle Monorail project, NETTE apod.

2.4.2 REST (Representational State Transfer), REST API

„ REST – je architektura rozhraní navržená pro distribuované prostředí. REST navrhnul a popsal v roce 2000 Roy Fielding (jeden ze spoluautorů protokolu http) v rámci disertační práce Architectural Styles and the Design of Network-based Software Architectures. V kontextu práce je nejzajímavější kapitola 5, ve které Fielding odvozuje principy RESTu na základě známých přístupů k architektuře. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). REST je tedy na rozdíl od známějších XML-RPC či SOAP orientován datově, nikoli procedurálně. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.“ [5]

REST implementuje čtyři základní metody známé pod označením CRUD, neboli Create, Retrieve, Update a Delete; tedy vytvořit, získat, aktualizovat a smazat. Tyto metody jsou implementovány pomocí odpovídajících HTTP metod.

- získávání zdrojů – pro získání zdroje se používá asi nejčastěji používaná metoda – GET. Aby bylo možné identifikovat data, má každý zdroj vlastní identifikátor (URI). Odesláním HTTP požadavku metodou GET na URI rozhraní budou vráceny požadované informace, např. seznam uživatelů. Data i pomocí tzv. query parametrů lze filtrovat či upravovat.
- Vytvoření nového zdroje – pomocí metody POST nebo PUT se pošlou informace na server. V tuto dobu ještě není znám URI záznamu. Jako odpověď od serveru se vrací právě tento identifikátor, pokud operace proběhla v pořádku.
- Aktualizace zdroje – pomocí metody PUT se odešlou data na konkrétní zdroj s konkrétním URI.
- Smazání zdroje – pomocí metody DELETE se odešle požadavek na konkrétní zdroj s konkrétním URI.

V praxi bývají občas problémy s vyvoláním metody PUT či DELETE. Tyto metody nebývají často implementovány. V těchto případech je nutné použít nějakou jinou, zástupnou metodu [6].

2.4.3 AJAX (Asynchronous JavaScript and XML)

AJAX, neboli Asynchronous JavaScript and XML, využívá asynchronní komunikaci za pomoci JavaScriptu. Srdcem komunikace je vytvoření požadavku skrze XMLHttpRequest Object. Požadavek je předán serveru, kde je zpracován a následně vrácen zpět na klienta, ve kterém je výsledek operace interpretován pomocí JavaScriptu uživateli.

V poslední době je AJAX často skloňovaný pojem. Jedná se o nástroj, který se do povědomí uživatelů dostal hlavně díky firmě Google (Gmail, YouTube, Google maps, …). AJAX umožňuje prohlížečům přijímat data na klientovi ze serveru bez nutnosti odesílat celý obsah stránky. Umožňuje tak vytvářet dynamické stránky, navozující pocit okamžité reakce.

Jednou z nejčastějších aplikací jsou našeptávací pole na webových stránkách. Téměř ihned poté, co jsou stisknuty klávesy, jsou stisky odesílány na server, kde jsou vyhodnocovány, a uživateli se zobrazí predikovaný text [7].

Technologie v pozadí Ajaxu, které jsou viděny jako důležité součásti ajaxového řešení jsou následující:

- *HTML/XHTML – hlavní jazyk pro reprezentaci obsahu webové stránky*
- *CSS – poskytuje stylistické možnosti pro formátování HTML/XHTML*
- *DOM – slouží pro dynamické změny v načtené stránce*
- *XML – formát pro výměnu dat*
- *XSLT – přetváří XML do XHTML*
- *XMLHttp – hlavní zprostředkovatel komunikace*
- *JavaScript – skriptovací jazyk použitý pro naprogramování engine Ajaxu*

Všechny tyto technologie jsou pro ajaxová řešení k dispozici, ale jenom tři z nich jsou skutečně nezbytnné – HTML/XHTML, DOM a Javascript. XHTML je nutné pro zobrazování informací na stránce, zatímco DOM je nezbytný pro změnu částí XHTML stránky bez jejího opětovného načítání. Poslední z nich JavaScript, je nezbytný pro zahájení spojení mezi klientem a serverem a manipulaci s DOM pro aktualizaci webové stránky. Ostatní technologie jsou pro vylepšení ajaxového řešení užitečné, ale nejsou nezbytné. [7]

AJAX je v dnešní době nedílnou součástí responsivních webových aplikací. Téměř si nelze představit, že by moderní aplikace nepoužívala AJAX. Našeptávací pole jsou jen jednou možnou aplikací. Mnohem častěji se dnes touto technologií získávají data ze serveru.

2.4.4 jQuery

Jedná se pravděpodobně o nejznámější JavaScriptovou knihovnu. Její použití je univerzální, není vázáno žádným nástrojem. Jeho největší silou je lehká

modifikovatelnost a široká fanouškovská základna. Díky tomu vzniká spousta zásuvných modulů, které ulehčují práci ostatním programátorům. Knihovna je užitečná hlavně ve sjednocení chování pro všechny prohlížeče. Není potřeba nadále vyvíjet prostředky podle závislosti na cílovém prohlížeči. Rozšiřuje standardní JAVASCRIPT EVENT model o další vlastnosti, největší síla se však projevuje při prohledávání DOM.

jQuery dnes nabízí i vlastní uživatelské rozhraní, které obsahuje komponenty, efekty a nástroje, které sjednocují chování komponent pro uživatele a ulehčují návrhářům implementaci jinak složitě programovatelných vlastností.

Pro návrháře cílené na mobilní platformy je na trhu i jQuery Mobile Framework, se kterým lze celkem jednoduchým způsobem vytvářet aplikace určené především pro mobilní telefony a tablety. Obsahuje komponenty optimalizované pro různá rozlišení. Výhodou je cílenost návrhu na responsivní design.

jQuery Mobile není jediný JavaScriptový framework. K dalším známým nástrojům patří i Sencha či Angular.

Sám o sobě JQuery není nástroj pro tvorbu aplikací jako takový, ale v kombinaci s REST a AJAX může pomoci vytvořit čisté HTML aplikace, které nebude třeba kompilovat.

2.4.5 Frontend frameworky

Mezi moderní metody vyvíjení aplikací patří tzv. frontend frameworky. Jsou to frameworky zaměřené – spíše než na obsahovou stránku – na stránku prezentace výsledku za využití prohlížeče. Poskytují nástroje k tvorbě stránek, bez starostí o rozložení prvků na stránce s ohledem na responsivitu stránky. [8]

Mezi zástupce tohoto typu nástroje by se dal považovat nástroj s názvem **Bootstrap**. Jedná se o open source framework, který byl uvolněn v roce 2011 a

vytvořen pro Twitter. Při tvorbě designu (front-endu) aplikace je využíván čistý HTML, CSS a JavaScript [9].

Za další zástupce tohoto druhu jsou považovány frameworky Foundation, Sencha či jQuery Mobile.

2.4.6 XML

XML je ve své podstatě formovaný text. Umožňuje nám definovat vlastní sadu značek, které chceme v dokumentu používat. Díky různým nástrojům a široké podpoře se dá XML využívat nejen k ukládání textu. V dnešní době je spousta nástrojů, které umožňují XML používat jako datové zdroje nebo jako prostředek k vytváření dynamických webových stránek. Hlavní výhodou XML je schopnost validace uložených informací. Navíc je XML formát dobře čitelný, a to nejen strojem, ale i člověkem. Nejedná se tedy jen o čistě strojový formát [10].

Jak již bylo řečeno, XML je vhodný na ukládání dat, protože umožňuje validaci. Validace je prováděna jak na úrovni formální, tak na úrovni obsahové pomocí DTD (Document Type Declaration) nebo XSD (XML Schema Definition).

DTD slouží k nadefinování platných elementů a atributů pro XML a k deklaraci vztahů mezi jednotlivými elementy. DTD lze umístit přímo do XML dokumentu, nebo jej lze umístit mimo. Umístění uvnitř souboru je značně nepraktické, v případě použití jinde je nutné deklaraci kopírovat.

„DTD se k dokumentu přidává pomocí deklarace typu dokumentu (DOCTYPE), která je umístěna na začátku dokumentu ihned za XML deklarací. Nejčastěji je DTD uloženo v samostatném souboru, aby mohlo být využíváno v mnoha dokumentech. V tomto případě má deklarace tvar:

<!DOCTYPE *kořenový_element* SYSTEM "*URL*">" [11]

Tím, že k dokumentu připojíme DTD, získáme možnost strojové kontroly obsahu dokumentu a navíc nám to při tvorbě dokumentu může nabídnout pomoc při sestavení správné struktury.

XSD vznikl v roce 2001 jako W3C (World Wide Web Consortium) doporučení. Ve své podstatě se jedná o alternativu k DTD. Oproti DTD však umožňuje definovat datové typy jednotlivých elementů či atributů. XSD se zapisuje pomocí XML. V rámci XSD lze využívat pro kontrolu i vyhledávacího nástroje XPath. Je tím umožněna např. kontrola jedinečnosti elementů.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="wikipedista">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="jmeno" type="xs:string"></xs:element>
            <xs:element name="prijmeni"
type="xs:string"></xs:element>
            <xs:element name="pocetEditaci"
type="xs:integer"></xs:element>
        </xs:sequence>
        <xs:attribute name="uid" type="xs:string"></xs:attribute>
    </xs:complexType>
 </xs:element>
 </xs:schema>
```

příklad definice XSD, zdroj Wikipedia

XPath (XML Path Language) je nástroj – dotazovací jazyk – stojící za téměř veškerou prací s XML. Umožňuje vyjádřit cestu od nějakého uzlu k jinému elementu nebo atributu. Může tedy připomínat adresářovou strukturu v souborovém systému. Výsledkem operace hledání může být jeden, více nebo žádný element či atribut [12].

XML není v dnešní době využíván jen pro skladování dat, stává se i mocným pomocníkem při tvorbě formátovaných výstupů. K tomu jsou určeny technologie XSL (eXtensible Stylesheet Language) a XSLT (XSL Transformation). J. Kosek popisuje XSL následujícími slovy:

„XSL je stylující jazyk upravující výstupy z XML pro výstup. Když vznikal jazyk XSL, měl to být skutečně jen starší bratříček kaskádových stylů (CSS). Umožňoval samozřejmě definovat vzhled jednotlivých elementů – způsob jejich zarovnání,

velikost a styl písma, barvy apod. Kromě toho jej šlo použít i k takovým věcem, jako je automatické generování obsahu, číslování obrázků, kapitol apod. Postupně se ukázalo, že XSL má sloužit ke dvěma poměrně odlišným věcem – k transformaci XML dokumentů a k definici vzhledu jejich formátování.“ [13]

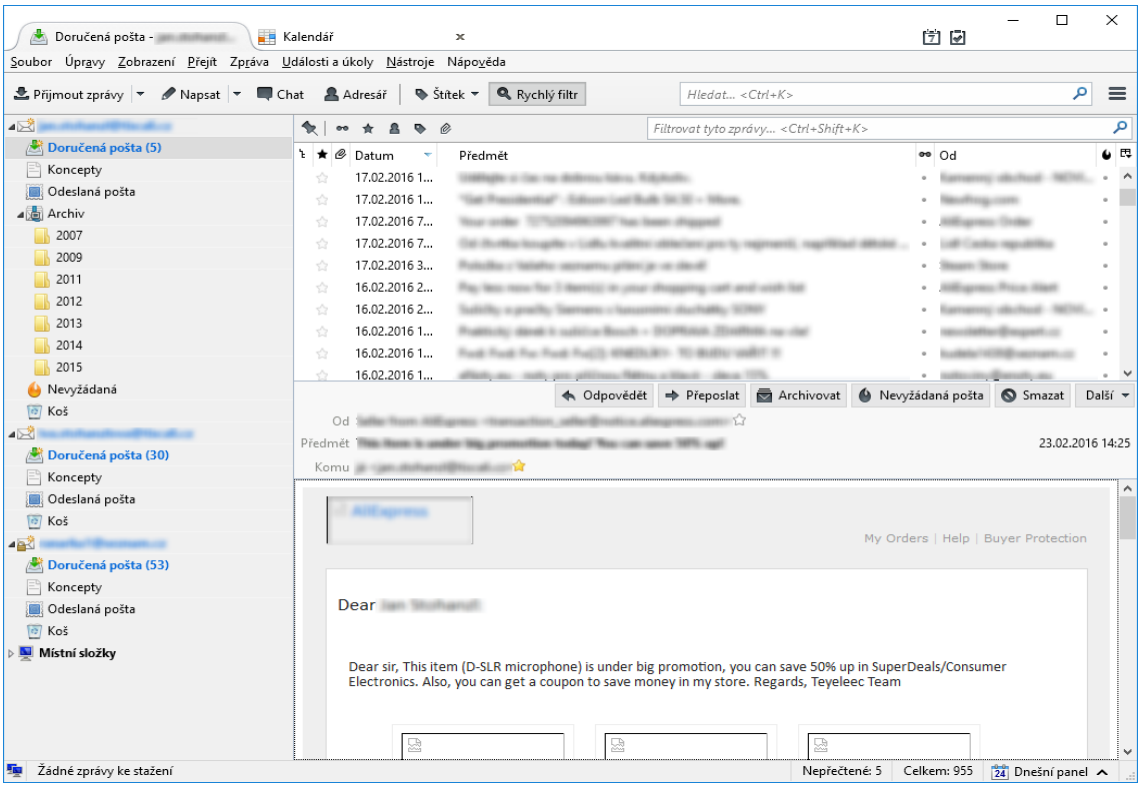
XSLT je v podstatě jazyk umožňující transformaci XML do jiné podoby (jiného XML). Byl navržen pro použití jako část XSL. K rozšíření XSL obsahuje XML slovník pro specifikaci formátování. XSL specifikuje stylování XML dokumentu použitím XSLT. XSLT popisuje, jak má být dokument transformován na jiný XML dokument používající formátovací slovník. XSLT je zároveň navržen k nezávislému užití na XSL. [14].

3 ANALÝZA

Tato kapitola se bude zabývat okolnostmi, které předcházejí vlastnímu návrhu frameworku. Nejprve je nutné stanovit si referenční cíle. K tomuto účelu slouží rozbor vzorové desktopové aplikace. V rámci rozboru budou vytyčeny potřeby uživatelů za pomoci USE CASE diagramů.

3.1 Rozbor vzorové desktopové aplikace

Jako vzor pro vytváření frameworku bylo použito třídílného rozdělení, které v dnešní době používá spousta aplikací. Jak je vidět na obrázku 1, okno je rozčleněno na nabídku (složky), řádkový přehled a detail. Mailový klient byl vybrán jako zástupce nejrozšířenější aplikace tohoto rozložení, které je vhodné nejen pro mailové klienty, ale i pro agendy.



Obrázek 1 – Vzorová aplikace Mozilla Thunderbird

3.1.1 Autentizace, autorizace

U každé moderní aplikace je kladen důraz na bezpečnost dat. Nelze tedy zobrazit chráněná data bez ověření, uživatel má právo tyto soubory vidět pouze v případě, že je mu to povoleno. Aby bylo možné tuto informaci zjistit, je nutné uživatele autentizovat, čímž se ověří totožnost uživatele. Samotným ověřením uživatele však ještě není zajištěno, že požadovaný obsah smí skutečně vidět nebo dokonce měnit. K tomuto účelu slouží autorizace, která umožní uživateli přiřadit uživatelskou roli vymezující viditelnost dat a operací nad nimi. Možnosti administrátora či řadového pracovníka se budou s nejvyšší pravděpodobností lišit.

V praxi nastávají situace, kdy je uživatelům umožněno zastávat více rolí. Pak je vhodné, aby měl takový uživatel možnost měnit přidělené role a aby měl vždy jasno, pod jakou uživatelskou rolí pracuje.

3.1.2 Seznam složek

V levé části okna jsou dostupné registrované e-mailové účty. Každý účet je členěn do složek, podle kterých je příchozí pošta rozdělována. Složky mohou být podle poskytovatele účtu rozdílné a mohou obsahovat i složky uživatelsky definované. Je nutné si uvědomit, že program je spuštěný v operačním systému pod autentizovaným uživatelem, zobrazené složky jsou tedy platné jen pro autorizovaného uživatele.

Pro lepší přehlednost jsou v programu zobrazovány u každé složky ikony a popisek identifikující obsah složky. Složky jsou seskupovány podle vlastníka a zobrazovány hierarchicky. Složky pod různými e-mailovými účty se stejným obsahem (např. doručená pošta, koncepty) jsou zobrazeny stejnou ikonou. Vybraná složka je opět zvýrazněna, aby bylo možné identifikovat příjemce zpráv.

Stejné by to mělo být i u agendy. Seznam složek by měl být přizpůsoben uživatelské roli, měly by být zobrazeny pouze takové složky, se kterými uživatel ve

vybrané roli smí pracovat. Složky, které k sobě logicky patří, by měly být seskupovány, srozumitelně popsány, aby bylo jasné, co je jejich obsahem, a pro lepší optickou orientaci by měly mít grafickou značku, která by mohla napovídat o obsahu.

3.1.3 Přehledové tabulky

Přehledové tabulky slouží pro rychlou orientaci mezi zobrazenými záznamy. Účelem přehledových tabulek je zobrazit stěžejní informace, podle kterých se koncový uživatel bude schopen orientovat. Zde u vzorové aplikace jsou za stěžejní informace považovány datum, čas, odesílatel a předmět zprávy.

U obecné agendy nelze jednoznačně stanovit stěžejní informace. Údaje zobrazené v přehledu by měly být vypovídající o obsahu vybrané složky. Opakující se atributy jsou pro přehled nevypovídající a nežádoucí. Uživatel by měl být schopen rychle vyhledat záznam, ke kterému se následně zobrazí podrobnější informace. Vybrané záznamy v tabulce bývají opticky zvýrazněné, aby bylo snadnější spárovat zobrazený detail se záznamem v tabulce.

Tabulka zobrazující informace je dynamická a její rozložení se přizpůsobuje zobrazeným informacím. V záhlaví tabulky jsou zobrazeny popisné informace sloupců vysvětlující obsah. Kliknutím na záhlaví lze záznamy třídit, a to vzestupně, nebo sestupně. V případě, že je zobrazeno více záznamů, než je možné zobrazit, je umožněno záznamy posouvat. Data mohou být zarovnána ve sloupci vlevo, vpravo nebo na střed. Bývá zvykem, že číselné položky jsou zarovnávány vpravo, obrázky a ovládací prvky bývají zarovnávány na střed, ostatní informace vlevo.

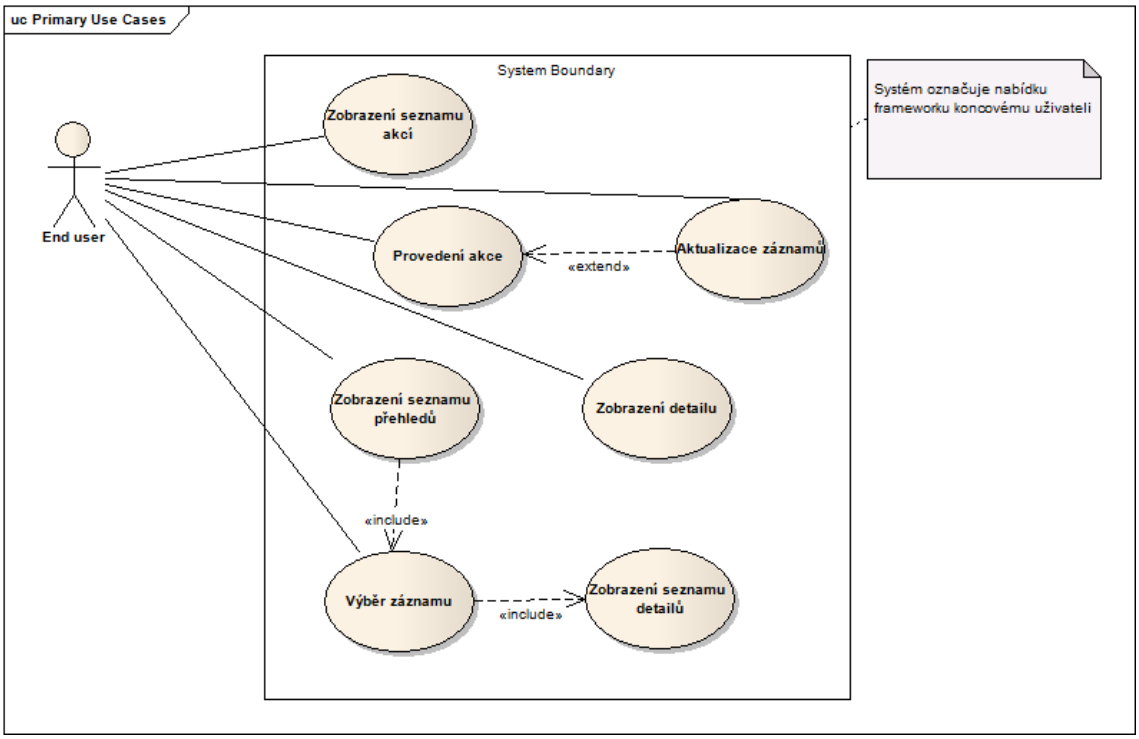
3.1.4 Detaily záznamu

Pro vybraný záznam je většinou požadováno zobrazit detailní informace. Ve vzorové aplikaci se detailem rozumí obsah vybrané zprávy. Pro tuto aplikaci se nepředpokládá jiný detail, ale dala by se zobrazit zpráva v surové podobě.

Aplikace pracující se zvolenou agendou nemusí mít k zobrazení pouze jeden detail. Informace mohou být členěny např. na obecné, technické, provozní či historii. V závislosti na vybraném záznamu v kombinaci s vybranou složkou by měly být zobrazeny informace o daném záznamu v patřičné oblasti.

3.2 Případy užití

Případy užití představují rozbor předpokládaného použití aplikace a specifikace úkonů, které se při běžném používání mohou vyskytovat. Používání aplikace může být vnímáno rozdílně z pohledu uživatelů, kterými mohou být v nejobecnějším pojetí administrátor, který konfiguruje webovou aplikaci, a koncový uživatel, který výsledek konfigurace bude používat.

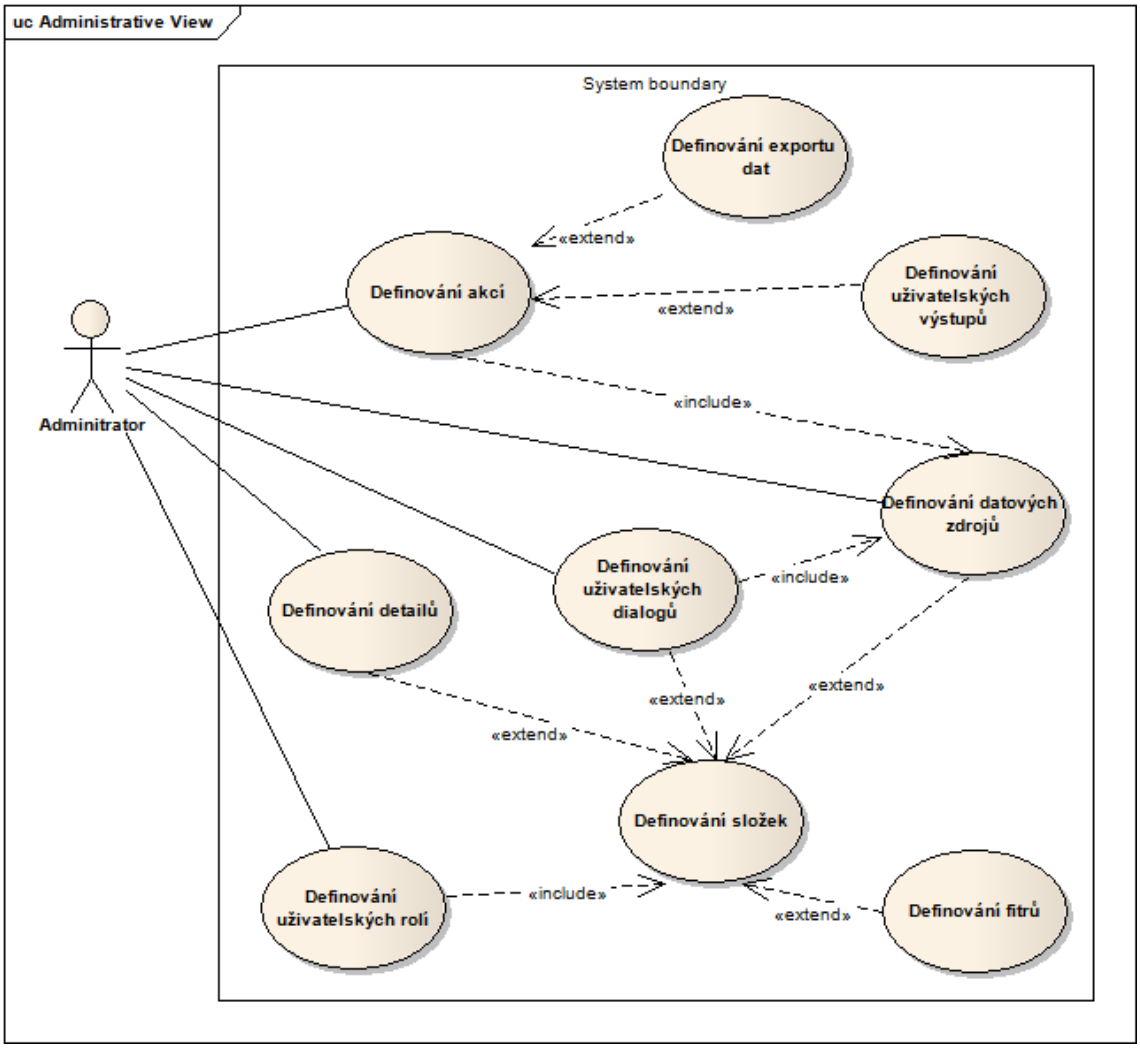


Obrázek 2 – Use case diagram koncového uživatele

Obrázek 2 zobrazuje základní požadavky na systém z pohledu koncového uživatele. Těmito požadavky jsou akce vycházející z analýzy ukázkové aplikace. Akce zde uvedené by měly být dostačující ke splnění funkčních požadavků na aplikaci vytvořenou frameworkem. K základním vlastnostem patří autentizace uživatele. Ostatní operace nemohou být spuštěny, aniž by uživatel byl systémem

autentizován. Díky autentizaci dochází i k načtení uživatelského profilu a zjištění dostupných uživatelských rolí uživatele.

Administrátor, který sestavuje výslednou webovou aplikaci, bude mít jiné potřeby oproti koncovému uživateli. Jeho pohled na systém je zaměřen hlavně na architekturu frameworku a stavbu cílové aplikace.



Obrázek 3 – Use case diagram administrátora

Jednotlivé pohledy odráží skutečnost různých potřeb uživatelů. Administrátoři jsou zde experti či klíčoví uživatelé, kteří se snaží koncovým uživatelům připravit aplikaci splňující jejich požadavky. Potřeby koncových uživatelů se různí, nicméně v obecné rovině jsou nároky na systém stejné. Patří

sem nutnost prohlédnout a upravit data, případně je zformovat do tiskového výstupu.

3.3 Scénáře pro koncového uživatele

V této kapitole budou podrobně popsány interakce koncového uživatele se systémem po jednotlivých akcích, jak jsou zobrazeny v USE CASE diagramu na obrázku 2.

3.3.1 Zobrazení seznamu přehledů (složek)

- Uživatel předá přihlašovací údaje systému
- Systém provede ověření uživatele
- Pokud systém nalezne více rolí, nabídne uživateli dialog s volbou zvolené role.
    - Uživatel si vybere roli, pod kterou se chce nahlásit do systému
    - Systém provede načtení uživatelského profilu, seznam dostupných přehledů pod danou rolí a přesměruje uživatele na stránku se zobrazeným přehledem.
- Jinak systém provede načtení uživatelského profilu, seznam dostupných přehledů pod danou rolí a přesměruje uživatele na stránku se zobrazeným přehledem
- Systém ukončí úlohu

3.3.2 Zobrazení seznamu detailů

- Uživatel provede výběr záznamu v přehledu
- Systém načte uživatelský profil
- Systém načte zdroj patřící vybranému přehledu
- Systém provede kontrolu uživatelské role
- Pokud role souhlasí
    - Systém spustí úlohu Kontrola dostupných rolí

    - Systém zobrazí seznam dostupných rolí
    - Systém ukončí úlohu
- Jinak systém vrátí prázdný seznam
- Systém ukončí úlohu

3.3.3 Zobrazení detailu

- Uživatel vybere záznam v přehledu
- Systém provede úlohu Zobrazení seznamu detailů
- Uživatel vybere detail k zobrazení
- Systém provede načtení zdrojů detailu
- Systém zpracuje požadavek
- Systém zobrazí uživateli požadovaný detail
- Systém ukončí úlohu

3.3.4 Zobrazení seznamu akcí

- Uživatel vybere záznam v přehledové tabulce
- Systém provede kontrolu uživatelské role
- Systém načte seznam akcí pro daný přehled
- Systém zobrazí přehled uživatelských akcí nad vybraným záznamem
- Systém ukončí úlohu

3.3.5 Provedení akce

- Uživatel provede výběr záznamu v přehledové tabulce
- Systém provede spuštění úlohy Zobrazení seznamu akcí
- Uživatel vybere zvolenou akci
- Systém provede načtení struktury akce
- Systém provede kontrolu spuštění
- Pokud kontrola nebrání spuštění
    - Provede se spuštění aktivního kódu akce
    - Systém ukončí akci

- Jinak systém zobrazí výsledek operace kontroly
- Systém ukončí akci

3.4 Scénáře pro administrátora

V této kapitole budou popsány interakce administrátora se systémem podle jednotlivých akcí, jak jsou zobrazeny v USE CASE diagramu na obrázku 3

3.4.1 Definování datových zdrojů

- Uživatel zvolí typ datového zdroje a název
- Systém provede registraci nového datového zdroje
- Systém nabídne uživateli formulář pro vyplnění atributů vybraného datového zdroje
- Uživatel vyplní zvolené atributy
- Systém provede kontrolu atributů, zdali jsou vyplněna požadovaná pole
- Pokud kontrola polí je v pořádku
    - Pak systém provede zápis atributů
    - Systém ukončí úlohu
- Jinak systém zobrazí chybové hlášení uživateli

3.4.2 Definování detailů

- Uživatel vytvoří nový detail
- Systém provede registraci detailu
- Systém načte seznam uživatelských dialogů
- Systém zobrazí seznam uživatelských dialogů
- Uživatel vybere uživatelský dialog, na kterém bude zobrazen detail
- Systém zapíše atribut k detailu
- Systém načte seznam dostupných datových zdrojů typu *ISimple*
- Systém zobrazí seznam datových typů
- Uživatel vybere datový zdroj, určený k získání informací detailu

- Systém zapíše informace
- Systém provede reload detailů
- Opakuj
    - Systém vloží typovou úlohu Definování uživatelských rolí
- Dokud uživatel nezvolí konec
- Systém ukončí úlohu

3.4.3 Definování uživatelských rolí

- Uživatel vytvoří novou uživatelskou roli
- Systém provede registraci nové role
- Systém nabídne uživatelský dialog na vytvoření role
- Uživatel doplní atributy
- Systém provede kontrolu jednoznačnosti role
- Pokud je kontrola v pořádku
    - Pak systém uloží změny
    - Systém provede reload zdrojů
    - Systém zobrazí zprávu uživateli o ukončení akce
    - Systém ukončí úlohu
- Jinak systém zobrazí chybový dialog

3.4.4 Definování složek

- Uživatel spustí akci Definování složek
- Systém provede načtení uživatelských rolí
- Systém zobrazí uživateli seznam rolí
- Uživatel vybere požadovanou roli
- Systém zobrazí formulář na vytvoření složky
- Uživatel vyplní požadované atributy a odešle formulář
- Pokud má být složka zobrazena jako přehled
    - Systém načte seznam datových zdrojů
    - Systém zobrazí seznam datových zdrojů typu *IStructured*
    - Uživatel vybere datový zdroj

    - Extension point: Definování filtru
    - Extension point: Definování detialu
    - Extension point: Definování datových zdrojů
    - Systém nabídne formulář pro definici primárního klíče
    - Uživatel vloží informaci o primárním klíči
    - Systém uloží informace o přehledu
    - Systém ukončí úlohu
- Jinak pokud má být složka zobrazena jako detailní záznam
    - Extension point: Definování datových zdrojů
    - Extension point: Definování uživatelských dialogů
    - Systém načte seznam uživatelských pohledů
    - Systém zobrazí seznam uživatelských pohledů
    - Uživatel vybere uživatelský pohled
    - Systém načte seznam datových zdrojů typu *ISimple*
    - Systém zobrazí seznam datový zdroj
    - Uživatel vybere datový zdroj
    - Systém provede uložení definice
    - Systém ukončí úlohu
- Jinak systém provede uložení definice
- Systém ukončí úlohu

3.4.5 Definování akcí

- Uživatel vytvoří novou akci
- Systém načte seznam tříd implementující rozhraní *IActionExecution*
- Systém načte seznam tříd implementující rozhraní *IActionAfter*
- Systém doplní číselníky načtenými seznamy
- Systém zobrazí uživatelský dialog vytvoření akce
- Extension point: Definování uživatelských výstupů
- Extension point: Definování exportu dat
- Uživatel potvrdí dialog
- Systém provede kontrolu

- Pokud kontrola proběhne v pořádku
    - Systém uloží akci
    - Systém provede reload akcí
    - Systém ukončí úlohu
- Jinak systém zobrazí chybové hlášení

3.4.6 Definování uživatelských dialogů

- Uživatel spustí typovou úlohu
- Systém načte seznam dostupných komponent
- Systém zobrazí formulář pro tvorbu uživatelského dialogu
- Uživatel nadefinuje vzhled, vlastnosti dialogu a odešle změny
- Systém provede kontrolu zadání
- Pokud kontrola proběhne v pořádku
    - Systém uloží uživatelský dialog
    - Systém ukončí úlohu
- Jinak systém zobrazí chybové hlášení

3.4.7 Definování filtrů

- Uživatel spustí typovou úlohu
- Systém načte seznam složek s přehledovou tabulkou
- Systém zobrazí seznam uživateli
- Uživatel vybere přehledovou tabulku, pro kterou chce vytvořit filtr
- Systém vytvoří filtr pro přehledovou tabulku
- Systém zobrazí dialog pro definici položek filtru
- Uživatel vyplní položky filtru
- Systém uloží filtr ke složce přehledové tabulky
- Systém ukončí typovou úlohu

3.4.8 Definování uživatelských výstupů

- Uživatel spustí typovou úlohu

- Systém načte seznam tříd implementující rozhraní *IActionExecution*
- Systém doplní číselník dostupných výstupů
- Systém zobrazí dialog pro nastavení výstupů
- Uživatel vyplní nastavení
- Systém uloží změny
- Systém ukončí úlohu

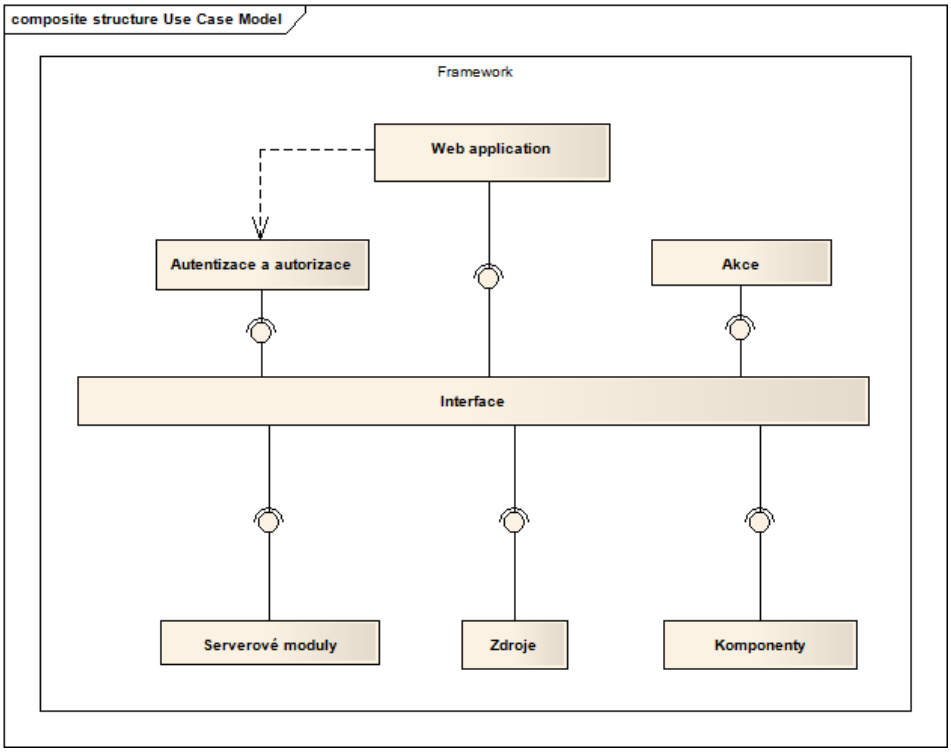
3.4.9 Definování exportu dat

- Uživatel spustí typovou úlohu
- Systém provede načtení dostupných formátů exportu
- Systém zobrazí dialog s nastavením exportu
- Uživatel provede nastavení exportu
- Systém provede uložení změn
- Systém ukončí úlohu

4 Architektura frameworku

Pro tvorbu frameworku bylo použito Microsoft Visual Studio 2015, programovací jazyk C#, .NET 4.0. Výsledná kompilace je publikována na serverech s operačním systémem Windows pod IIS (Internet Information Services).

Framework je rozdělen do několika základních modulů, které jsou vzájemně více či méně provázány. Většinou se jedná o nezávislé aplikace (dynamické knihovny) spojené jednou webovou aplikací. Některé moduly lze rozšířit zvenčí, jiné je nutné upravovat přímo v assembly.



Obrázek 4 – Architektura frameworku

Jak je vidět na obrázku 4, styčným bodem frameworku je modul interface. Aplikace se snaží využívat reflexe, aby nebylo nutné odkazovat se na konkrétní třídy. Takto se volání omezuje pouze na vlastnosti interface, celkem bez větších obtíží lze dynamicky rozšiřovat schopnosti, základnu komponent či serverové moduly apod. V tabulce níže jsou jednotlivé moduly vysvětleny.

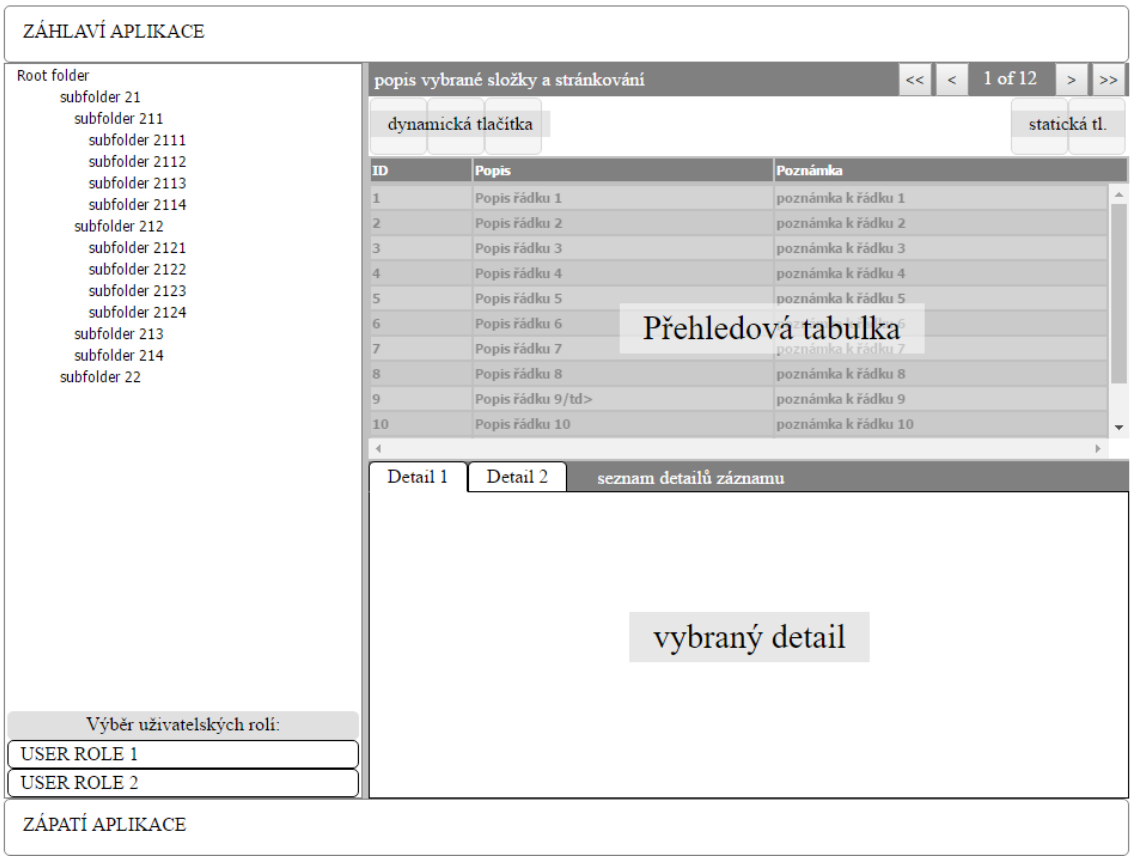
Tabulka 2 – Moduly frameworku

Název modulu	**Stručný popis**
Interface	deklarace společných rozhraní
Autentizace a autorizace	ověření uživatelů a přístupu k operacím
Uživatelské role	definice uživatelských rolí
Akce	definice povolených akcí
Serverové moduly	moduly zpracovávající požadavky volané klientem
Komponenty	komponenty pro tvorbu dialogových oken

4.1 Grafické rozhraní

Návrh grafického rozhraní byl proveden dle vzorové aplikace. Na obrázku 5 je vidět rozložení vytvořeného aplikačního rámce frameworku. Jsou zde k vidění jednotlivé moduly popsané v tabulce 2, ze kterých je framework sestaven. Jednotlivé části modulů se vzájemně prolínají a doplňují. Nemusí být používány jen samostatně.

Framework staví z modulů fungující celek. V uživatelských volbách jsou definovány role a dostupné složky. Každá složka má definovány zdroje, detaily a dynamická tlačítka nad záznamy. Dynamická tlačítka volají akce, které mohou vyvolávat dialogová okna sestavená z komponent.



Obrázek 5 – Wireframe diagram aplikace

4.2 Interface

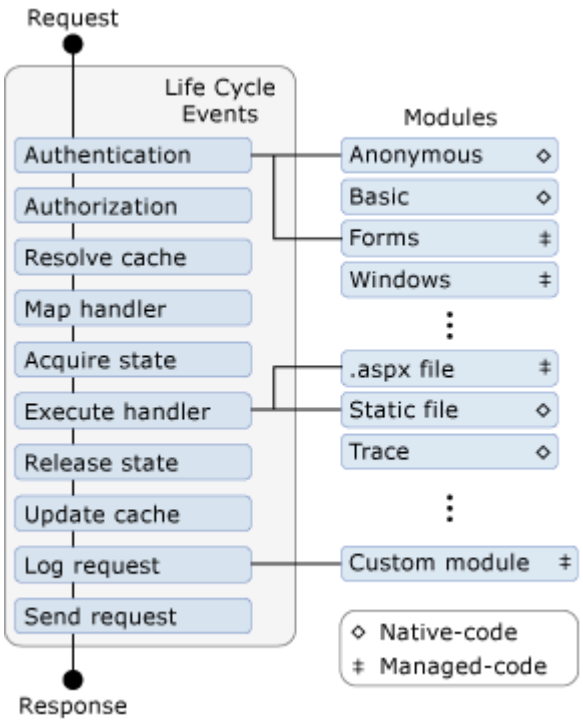
Interface (rozhraní) zajišťuje spolupráci mezi jednotlivými moduly aplikace a umožňuje rozšiřovat aplikaci o podpůrné funkce či komponenty. Některé moduly využívají reflexe Microsoft .Net. Při startu webové aplikace dojde k prozkoumání všech dostupných assemblies a přezkoumá se, zda jsou přítomny třídy implementující zájmová rozhraní. Jsou-li zde dostupné hledané třídy, dojde k jejich zaregistrování a vytvoření instancí těchto tříd. Instance tříd držících informace o dostupných implementacích jsou vytvářeny dle návrhového vzoru *„Jedináček“*. Všechny vytvořené instance třídy se odkazují na jeden obsah. Tyto instance se chovajíjako statický objekt.

Instance implementovaných tříd jsou schopny za běhu vytvářet své vlastní instance s nastavením vyčteným z konfiguračních souborů. Třídy implementují rozhraní, které obsahuje odkaz na metodu *CreateObject*, čímž odpadá nutnost

přímého volání metody, jen se zavolá metoda rozhraní a nový objekt se vytvoří podle požadavků. Aktivují se pak až při vlastním použitím. Všechny objekty by měly být „*thread-safe*“, protože ve webových aplikacích může díky paralelnímu zpracování docházet k volání stejné třídy v jeden okamžik. Pokud by třídy nebyly „*thread-safe*“, docházelo by k ovlivňování výsledku operací, což by bylo pro práci frameworku nežádoucí.

4.3 Autentizace a autorizace

Autentizace ve frameworku je založena na .NET Forms autentizaci. Je vytvořený aplikační modul, který provádí kontrolu každého požadavku směřujícího do aplikace a provádí kontrolu, zdali uživatel vytvářející požadavek má na tuto operaci oprávnění.



Obrázek 6 – Zpracování HTTP požadavku pomocí ASP.NET, zdroj [15]

Jak je vidět na obrázku 6, standardní HTTP požadavek prochází autentizací, při níž jsou spouštěny externí moduly. Díky tomu lze „přizpůsobit“ chování standardního procesu ASP.NET autentizace. Standardní autentizace spočívá ve

vyhledání *autentizační cookie*. Pokud není nalezena, dojde k přesměrování požadavku na výchozí stránku.

Aby bylo možné přizpůsobit standardní chování ASP.NET Forms autentizace, je zapotřebí při autentizaci uživatele vytvořit vlastní autorizační cookie. Poté, co je některou z dostupných metod uživatel ověřen, dojde k načtení uživatelského profilu, informace jsou serializovány, následně zašifrovány a uloženy do těla cookie. Tím jsou tyto informace dostupné při každém dotazu na server a dochází k úspoře času, který by byl nutný pro získání těchto informací.

Pro účely frameworku byly vytvořeny třídy implementující rozhraní *IFrameworkPrincipal* a *IFrameworkIdentity*, které jsou schopny informace z cookie dešifrovat a deserializovat. Ve frameworku se pak na tyto objekty lze odkázat a pracovat s nimi. V rozhraní *IFrameworkPrincipal* jsou informace o dostupných uživatelských rolích, jméno uživatele a odkaz na *IFrameworkIdentity*, kde lze vyčíst uživatelský profil.

4.4 Uživatelské volby (složky)

Uživatelské složky jsou v podstatě všechny dostupné volby uživatele pod právě přihlášenou rolí. Veškeré volby jsou zobrazeny v levém sloupci aplikace. Výběrem složky a záznamu jsou uživateli dostupné akce, které smí pod daným přihlášením provádět. Složky jsou zobrazeny ve stromové struktuře, aby bylo možné je členit. Složkami je uživateli umožněno provádět interakce s aplikací. Je zde shrnuto vše, co může framework uživateli pod danou konfigurací nabídnout. Od vybrané složky se odvíjí ostatní operace.

Každé složce lze nastavit základní atributy, jimiž jsou jméno, popis a styl. Ve složce je odkaz na definovaný zdroj dat. Pokud se očekává nadměrné množství dat, lze nastavit stránkování, omezení časovým intervalem, popřípadě nastavit výchozí hodnoty filtru a omezit tak rozsah zobrazovaných dat. Pro zvolený zdroj dat lze nadefinovat uživatelský filtr a seznam detailů, které se pro vybraný záznam přehledové tabulky mají zobrazit. Aby bylo možné nějakým způsobem provádět

akce nad záznamy, je zde možnost nadefinovat dynamická akční tlačítka, jimiž uživatel může iniciovat akce vedoucí k úpravě dat či vytvoření koncového výstupu.

4.4.1 Uživatelské filtry

Filtry jsou volně deklarovatelné v rámci konfiguračního souboru. Dle daného zdroje lze v rámci filtrů nastavit, které informace jsou dostupné k filtrování, jakým způsobem se má daný filtr zobrazit, popřípadě jakou metodou filtrovat.

- Number – filtrování sloupce jako číslo. U čísel je zobrazeno roletové menu, s volbou matematické operace porovnání číselné hodnoty se sloupcem (<, >, =,...).
- Date – filtrování datumových položek. Jsou zobrazeny dvě kalendářová pole a podle vyplněné hodnoty od–do je sestaven odpovídající filtr.
- String – u textového pole je porovnávání čistě jen text, nicméně lze nastavit, aby filtrování ignorovalo diakritiku, čímž zvýší citlivost filtru. Vyhledávání je case-sensitive – záleží na velikosti písmen.
- UpperLike – uživateli se nabídne možnost vyhledávání hledaného řetězce v požadované pozici v textu (začíná, obsahuje, končí). U tohoto filtru lze použít zástupné znaky (_ a %), popřípadě jejich alternativy z MS-DOS (? a *). Prohledávání probíhá SQL metodou **like** a nezáleží na velikosti písmen. Stejně jako u předchozího filtru lze zrušit diakritiku.
- ListOfValues – seznam hodnot; metoda filtrování spočívá v zobrazení roletové nabídky, jejíž obsah odpovídá sdruženým hodnotám v prohledávaném sloupci. Zobrazené hodnoty lze pomocí vyhledávacího řetězce filtrovat. Vzhledem k použití hodnot filtr využívá operaci rovnost.

4.4.2 Akční tlačítka

V uživatelských volbách se nastavují i tlačítka aplikace. Tlačítka jsou ve frameworku buď statická, nebo dynamická. Pokud jsou statická, pak jsou zobrazena vpravo nad přehledem vybrané složky, zpravidla jsou to tlačítka

filtrování záznamů, vytváření nových záznamů, export dat nebo uživatelské sestavy.

Dynamická tlačítka jsou zobrazována vlevo nad přehledem a jejich zobrazení je závislé buď na ověřovací funkci, nebo na složkách.

Obrázek 7 – Akční tlačítka frameworku

Na obrázku 7 je vidět rozložení tlačítek na stránce. Počet tlačítek se liší podle nastavené konfigurace a vybrané složky.

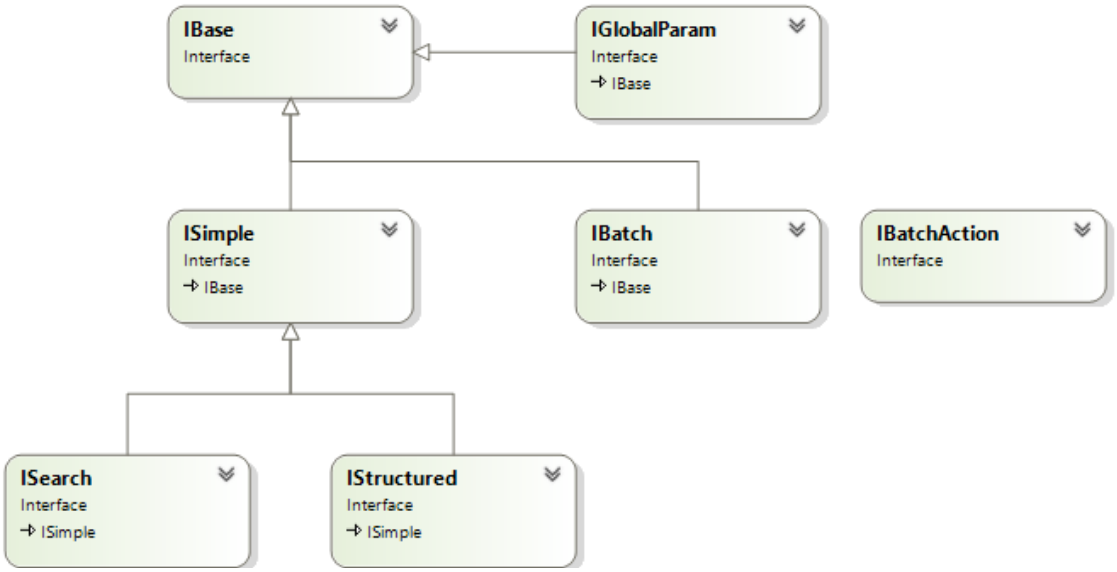
4.4.3 Detailní záložky

Pro každý záznam lze zobrazit jeho detail. Pro tyto účely slouží v nastavení složky, ve kterých je definován zdroj pro získávání informací a okno, ve kterém se daná informace má zobrazit. Každý detail je identifikován názvem, který by měl odpovídat obsahu, který zobrazuje. Zobrazení záložek závisí na způsobu definice. Buď jsou deklarovány obecně v rámci vybrané role a zobrazení jednotlivých záložek závisí na konfigurační hodnotě, nebo jsou definovány přímo u adresáře.

Pokud jsou deklarovány obecně, je jejich zobrazení závislé na výběrové funkci a mohou být zobrazeny jen ve specifických případech. Pokud jsou definovány u adresáře, jsou zobrazeny u tohoto adresáře vždy. Pokud je vždy zobrazen detail záznamu nad danou složkou, bez omezení, lze jej jednoduše definovat přímo u složky. Například u záložek historie nemusí být žádoucí, aby byly viditelné u nových záznamů. Pak je nutné jejich zobrazení řídit na úrovni záznamu.

4.5 Datové zdroje

Datové zdroje (resources) jsou srdcem celého frameworku. V tomto modulu se vyskytují všechny třídy související se získáváním či upravováním dat. Hlavní třída *GFResources* je napsaná dle návrhových vzorů Factory a Singleton. V rámci frameworku je vždy jen jedna instance této třídy a sdružuje všechny dostupné funkce spojené s používáním zdrojů. Třídy používané v rámci resources jsou děleny na dva druhy – na třídy získávající data a na třídy zpracovávající data; oba druhy musí implementovat základní rozhraní *IBase*. Strukturu závislostí jednotlivých rozhraní lze lépe pochopit z obrázku 8.



Obrázek 8 – Závislost rozhraní zdrojů, zdroj vlastní

Pro zobrazení dat jsou k dispozici tři rozhraní, která specializují *IBase*. těmito rozhraními rozumíme *IStructured* , *ISimple* a *ISearch*. Každé rozhraní má svůj význam pro framework a určuje, jakým způsobem bude s daty nakládáno, či spíše jaká data můžeme při práci očekávat.

Pro zpracování dat jsou zde pouze dvě rozhraní, a to *IBatch* a *IBatchAction*. *IBatch* je vlastně obálka udržující seznam tříd manipulujících s daty.

4.5.1 ISimple

Rozhraní *ISimple* se používá pro získávání pouze jednoho řádku dat. Je využíváno k získávání dat pro detaily nebo výchozí či aktualizační hodnoty formulářů.

Tabulka 3 – Popis definice rozhraní ISimple

Metoda/vlastnost	Výsledek	Druh	Popis
select	string	vlastnost	definice SQL příkazu
GetSqlCommand	string	metoda	vrací zpracovaný SQL příkaz
GetData	IDataObject	metoda	vrací výsledek provedeného SQL příkazu
UsedParams	List<string>	metoda	seznam použitých proměnných v SQL příkazu

4.5.2 ISearch

Rozhraní *ISearch* se využívá pouze pro roletová menu. Roletová menu lze ve frameworku používat jako strom. Očekávají se pouze tři sloupce, a to ID, popis a ParentID. ID slouží k jednoznačnému identifikování číselníkové hodnoty, v popisu se nachází zobrazovaná uživatelsky čitelná hodnota a ParentID, pokud není null, odkazuje na ID předka.

Pro zobrazení záznamů se předpokládá, že se vyskytují složitější číselníky, kde se nachází více hodnot, což z uživatelského hlediska bývá nepřehledné a s množstvím záznamů roste doba načítání číselníků. Proto se předpokládá, že záznamy bude třeba odfiltrovat – nad sloupcem, který má být použit pro výběrovou podmínka se použije proměnná **SEARCH**. Např.

select id, desc, null notree from table where desc like '%'||:search.

Rozhraní specializuje rozhraní *ISimple* vlastností searchText a přetěžuje metody GetData a GetSqlCommand. V těle implementující třídy se tedy vyskytují dva výběrové SQL dotazy – jeden slouží k načtení počátečních hodnot bez omezení, druhý ke zobrazení filtrovaných záznamů.

4.5.3 IStructured

Rozhraní *IStructured* je využíváno pro zobrazování záznamů v tabulkách a – stejně jako *ISearch* – specializuje rozhraní *ISimple*. Struktura rozhraní, jak název napovídá, je strukturovaná – rozděluje SQL příkaz na jednotlivé součásti, tzn. SELECT, FROM, WHERE a ORDER. Vychází se z faktu, že SQL dotaz lze napsat jak jednoduše, tak i složitě, a pro různé databázové platformy se může drobně lišit. Struktury SQL jsou často víceúrovňové a nelze jednoduše získávat jednotlivé části SQL příkazu. K tomu by se musel sestavit SQL parser, jenže – s přihlédnutím k faktu, že zdrojem dat nemusí být jen SQL datababáze – by mohlo dojít k chybnému rozdělení příkazu, což je nežádoucí.

Tabulka 4 – Popis definice rozhraní IStructured

Metoda/vlastnost	Výsledek	Druh	Popis
from	string	vlastnost	klausule from SQL příkazu
where	string	vlastnost	klausule where SQL příkazu
order	string	vlastnost	klauzule order SQL příkazu
count	string	vlastnost	SQL příkaz pro spočtení záznamů
GetCount	IDataConvertor	metoda	vrací počet záznamů instance třídy
columns	DataColumn[]	vlastnost	seznam sloupců ke zpracování

V přehledových tabulkách je často nutné záznamy třídit a filtrovat. K tomuto účelu je **nutností** znát strukturu SQL, aby bylo možné dotazy správně sestavit a přizpůsobit aktuálnímu požadavku. Dále rozhraní obsahuje možnost sestavit vlastní SQL příkaz k počítání záznamů. Pokud není uveden, provede se obalení původního SQL příkazu a spočítají se záznamy. Tato metoda je účinná, ale pro přehledy se často používají komplikované vazby či funkce, které nejsou omezujícího charakteru, ale mají zásadní vliv na rychlost spočítání záznamů. Proto je někdy lepší si napsat vlastní, jednodušší SQL příkaz počítající záznamy.

Vlastnost *columns* je využívána při sestavování zobrazované tabulky. Má zásadní vliv na způsob zobrazení dat. SQL sloupec lze přejmenovat, určit datový typ, což má vliv na formátování dat. V tabulkách se mohou vyskytovat i HTML fragmenty, u kterých se využívá vizuálního zobrazení záznamu (stavy,…). Aby při exportu nebyly vidět tyto HTML fragmenty, je zde možnost přesměrování na jiný sloupec obsahující jen hodnotu pro export.

4.5.4 IBatch

Rozhraní *IBatch* slouží jako obálka pro třídy implementující rozhraní *IBatchAction*, navíc drží informace o všech dostupných třídách, které toto rozhraní implementují. Vzhledem k tomu, že je možné vytvářet vlastní třídy, tento seznam je dynamický a pro různé aplikace nemusí být stejný. Konfigurační soubory neřeší, z které DLL knihovny přichází které třídy, ale ve výsledku musí být dávka zpracována. Jednotlivé akce jsou načteny při startu aplikace a pro další zpracování předávány skrze rozhraní implementujícím třídám.

Tabulka 5 – Popis definice rozhraní IBatch

Metoda/vlastnost	Výsledek	Druh	Popis
actions	IBatchAction[]	vlastnost	seznam vlastních akcí dávky
supportedActions	IBatchAction[]	vlastnost	seznam použitelných akcí dávek
Process	IDataSimpleReturn[]	metoda	Provádí zpracování dávky. Výsledkem je pole odpovídající maximální velikosti seznamu akcí s výsledkem operace.

Dávka se skládá z jednotlivých akcí, které jsou zpracovávány v pořadí, ve kterém jsou uvedeny v konfiguračním souboru. Výsledkem operace je buď úspěch, nebo návratové chybové hlášení. V případě, že jedna z akcí selže, je spuštěna metoda Rollback rozhraní *IDatabase*, která by měla zrušit provedené změny spuštěné dávky.

4.5.5 IBatchAction

Rozhraní *IBachAction* slouží k identifikaci třídy, provádějící manipulaci s daty frameworku.

Tabulka 6 – Popis definice rozhraní IBatchAction

Metoda/vlastnost	Výsledek	Druh	Popis
TAG	string	vlastnost	název XML elementu v konfiguračním souboru; pro jednotlivé třídy by měl být jednoznačný
executeIfColumn	string	vlastnost	podmíněné spuštění – sloupec
executeIfOperation	ExecuteOperation	vlastnost	podmíněné spuštění – operace
executeIfValue	string	vlastnost	podmíněné spuštění – hodnota
initialize	void	metoda	inicializace instance třídy
Process	IDataSimpleReturn	metoda	vlastní vykonání operace
CreateObject	IBatchAction	metoda	vytvoření nové instance třídy
GetXML	string	metoda	použití pro zpětnou serializaci

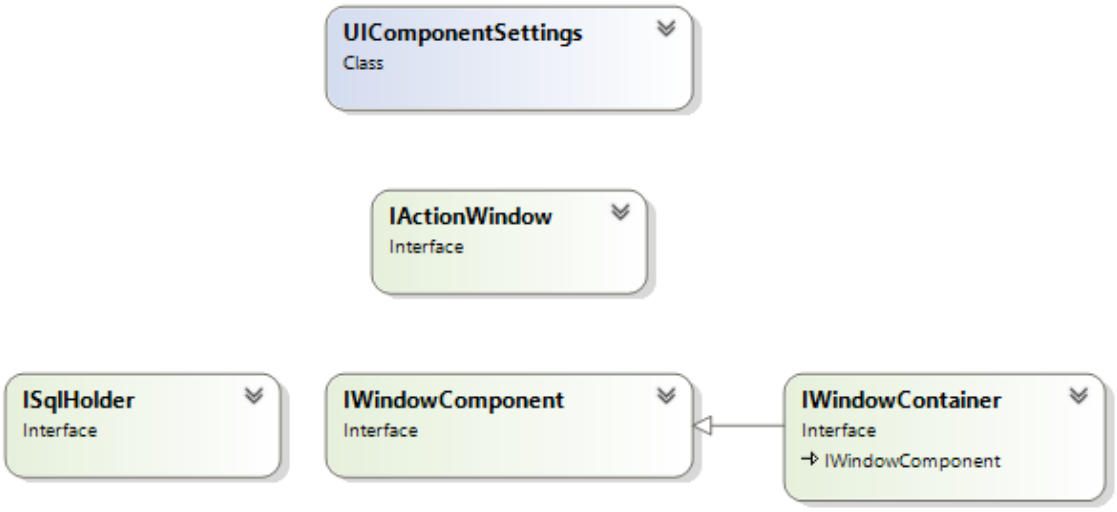
Vykonání požadované operace může být podmíněno, pokud jsou vyplněny „execute“ atributy. V případě, že je podmínka splněna, dojde k provedení kódu, nebo naopak. Metoda initialize je automaticky spouštěna po vytvoření objektu a umožňuje spuštění inicializačního kódu.

4.6 Komponenty

Modul komponenty v sobě skrývá prostředky, kterými framework komunikuje s koncovými uživateli. Jsou zde k dispozici veškeré prostředky potřebné k vytváření uživatelských dialogů.

Jádrem komponent je třída *UIComponentSettings* vytvořená dle návrhového vzoru Singleton a Facade. Při inicializaci třídy dojde k procházení Assembly a vyhledávání třídy implementující rozhraní *IActionWindow* a *IWindowComponent*. Rozhraní *IWindowComponent* slouží k identifikaci stavebních prvků pro akční

okna. Aby bylo možné vytvářet akční okna, je nutné nejprve zjistit seznam komponent. Ty jsou pak předávány jako parametry při vytváření instancí tříd implementujících *IActionWindow*. Rozhraní *IActionWindow* slouží k identifikaci komponent vystupujících jako akční okna. V tuto chvíli framework obsahuje pouze jednu třídu s tímto rozhraním.



Obrázek 9 – Třída UIComponentSettings, zdroj vlastní

Standardně jsou okna spouštěna pod autorizací, jsou však případy, kdy lze povolit přístup k oknu bez autorizace (např. registrace nového uživatele, zobrazení informací o stavu požadavku)

Stěžejní metodou *IActionWindow* je metoda prepare, která vrátí ve třídě implementující IWindowPrepared HTML a případně javascriptovou deklaraci. Vzhledem k tomu, že ve výsledku jsou komponenty renderovány v HTML stránce, jsou deklarace vlastností komponent frameworku velmi podobné jazyku HTML a přímo z nějvycházejí.

Jednotlivé komponenty pro stavbu oken se dají shrnout do dvou druhů: komponenta (*IWindowComponent)* a kontejner (*IWindowContainer)*. *IWindowComponent* jsou prvky, které jsou rovnou vykresleny, nemohou obsahovat další prvky.

Tabulka 7 – Seznam komponent typu komponenta

Název	Popis
combo	roletová nabídka
text	textové pole
memo	víceřádkové textové pole
radio	radiobutton
check	check box
dbTable	tabulkový přehled
date	datumové pole
image	obrázek
button	tlačítko
hidden	skryté pole
html	čistý HTML kód, nevázaný kód, umožňující zobrazit informace, které standardními komponentami frameworku nelze interpretovat
label	popisek

IWindowContainers specializují rozhraní *IWindowComponent* a jak název napovídá, jedná se o prvky obsahující jiné prvky. Vykreslování může být rozdílné od základní komponenty a může být parametricky řízené.

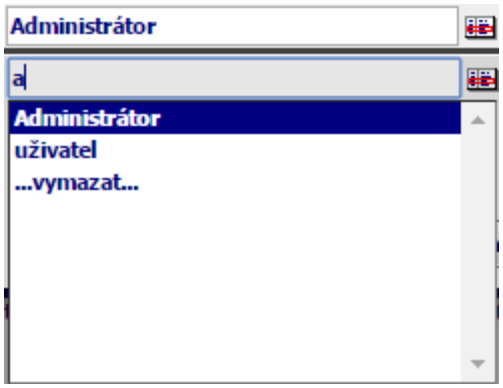
Tabulka 8 – Seznam komponent typu kontejner

Název	Popis
Layout	základní stavební kámen okna
Panel	ve své podstatě jen čistý HTML element DIV
TabControl	kontejner obsahující záložky
Table	definice tabulky
Fieldset	HTML element fieldset – pro sdružování prvků na formuláři

Framework předpokládá, že jako první komponenta na formuláři musí být umístěna komponenta typu kontejner.

4.6.1 Combo

Pro práci s daty je občas nutné nabízet uživateli pouze omezený seznam hodnot. K těmto účelům by mohl za určitých okolností sloužit HTML prvek Combo, nicméně s tímto prvkem by byla práce omezena pouze na data uložená přímo v seznamu. Při pokusu o integraci dynamického chování (načítání dat na vyžádání pomocí technologie AJAX) došlo k problémům s výměnou dat. Z tohoto důvodu byla napsána vlastní komponenta, která je schopna pracovat ve dvou režimech. Jedním režimem je volba z číselníku (není umožněno psát), druhá možnost je vyhledávání v číselníku. Pokud není v textu nic napsáno, je spuštěn dotaz, který vrací všechna data. V případě, že je uplatněn výběr, spouští se dotaz uvedený v *ISearch*, kde se v SQL příkazu nahrazuje parametr **[SEARCH]**.

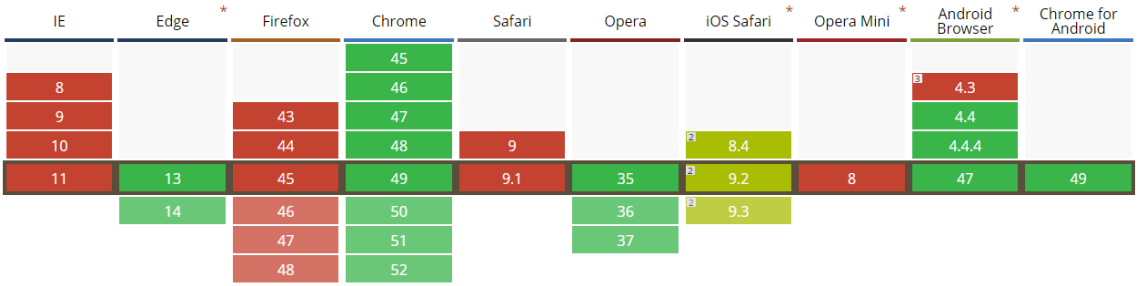


Obrázek 10 – Komponenta Combo

Na obrázku 10 je zobrazen zavřený seznam s vybranou hodnotou a seznam otevřený ve vyhledávacím módu.

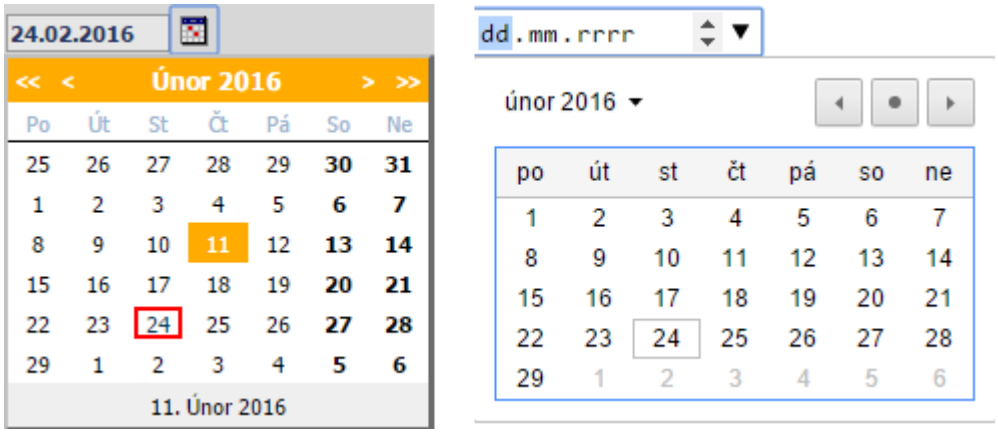
4.6.2 Date

Komponenta Date slouží, jak již název napovídá, k výběru data. Oproti komponentě Combo žádný standardní prvek typu date v HTML není. V HTML 5 se sice nachází prvek input typu date, ale ačkoliv podpora HTML 5 formulářových prvků funguje téměř ve všech soudobých prohlížečích, s podporou pole DATE je to horší. Dle serveru *http://caniuse.com* toto pole nepodporuje skoro polovina prohlížečů (viz obrázek 11).



Obrázek 11 – Podpora pole input typu date, zdroj [21]

Z tohoto důvodu bylo nutné vytvořit komponentu, která by měla výběr umožnit ve všech prohlížečích a sjednotit chování. Je potřeba brát zřetel na to, že datum je jeden z nejožehavějších datových typů. Pro každé národní prostředí může být datum interpretováno různým formátem – dle národních zvyklostí. Např. v Čechách je zvykem zobrazovat datum ve formátu *„dd.mm.yyyy“*, kde *dd* označuje den, *mm* měsíc a *yyyy* rok. V USA se ovšem datum skládá jiným způsobem – využívá se masky *„yyyy/dd/mm*“. Datum na klientovi by se mělo zobrazovat podle zvoleného nastavení.



Obrázek 12 – Komponenta Date

Na obrázku 12 je vlevo zobrazena komponenta Date s vybraným datem a otevřeným výběrovým dialogem v českém nastavení, vpravo pak input pole typu date zobrazeno v Chrome s otevřeným dialogem pro výběr data.

4.6.3 Label

Label znamená popisek. Každá aplikace potřebuje popsat jednotlivá pole. K tomuto účelu je používána komponenta Label. Ta je pro zobrazení vybírána dle způsobu chování. Pokud je popisek svázán s nějakým polem, pak je zobrazen v HTML jako standardní prvek label. Výhodou je, že po kliknutí na popisek dojde k aktivaci svázaného pole (vhodné např. pro checkbox nebo radio). Pokud není popisek svázán s políčkem, je zobrazen jako prvek span.

4.6.4 Hidden

Hidden je skryté pole. Používá se k přenosu informací, jako jsou identifikátory, dodatečné informace apod. Tato pole nejsou uživatelsky modifikovatelná, slouží pouze návrhářům. Komponenta je reprezentovaná standardním HTML polem input typu hidden.

4.6.5 Text a memo

Toto je nejčastěji používaná editační komponenta. Vychází jako ostatní komponenty ze základního HTML prvku input typu text. Nabízí uživateli možnost nastavení chování pole. Pokud je pole vyžadováno, lze nastavit uživatelskou zprávu spolu se zvýrazněním pro případ, že položka není vyplněna. Umožňuje omezení počtu znaků nebo kontrolu rozmezí hodnot, pokud se očekává hodnota typu číslo. Pole na vstupu ke zpracování prochází kontrolou, a pokud nevyhovuje typ nebo některá z omezujících podmínek, framework zareaguje na klientské straně hlášením, zvýrazní nevyhovující pole a nastaví do pole kurzor.

Memo je oproti textu komponenta, u které se předpokládá použití pro víceřádková vstupní data, proto je umožněno definovat počet řádků a sloupců. Memo je reprezentováno v HTML pomocí pole textarea.

4.6.6 Radio a check

Radio (radiobutton) slouží k výběru jedné volby z několika, checkbox neboli zaškrtávací pole slouží k výběru žádné možnosti, nebo všech. Obě komponenty

jsou odvozeny z HTML input typu checkbox nebo radio. U zaškrtávacích polí je problém, že jsou přenášeny hodnoty pouze vybraných možností. Je pak nutností zvolit si výchozí hodnotu pro případ, že položka není vybraná.

4.6.7 Table

Komponenta Table je komponenta typu kontejner. Obsahuje tedy další komponenty. Vychází z HTML prvku TABLE a je primárně určena pro vymezení rozložení komponent na stránce. Obsahuje řádky a sloupce, které mohou být slučovány pomocí atributů rowSpan a colSpan. Přesněji řečeno, tabulka obsahuje řádky a řádky obsahují sloupce.

V řádcích tabulky je jeden podstatný rozdíl oproti prvku HTML. Řádek tabulky obsahuje atributy řízení viditelnosti pomocí proměnných frameworku. Tím se stává užitečná např. při tvorbě formuláře na úpravu dat. Řízením viditelnosti lze zobrazit prvky, které mají být viditelné pouze při tvorbě záznamu, a naopak – skrýt prvky při editaci, které mají být viditelné jen při úpravě záznamu.

Tato komponenta není určena k zobrazení přehledů. K tomuto účelu je navržena jiná komponenta – DbTable.

4.6.8 DbTable

Tato komponenta je vytvořena pro účely prezentace víceřádkových dotazů neboli přehledů. Komponenta je přímo svázána se strukturovaným zdrojem dat. Dotaz může obsahovat podmínku, kterou je objem dat redukován pouze na vybranou položku ve složce či jinou proměnnou známou při vyvolání formuláře. Na rozdíl od předchozí komponenty tato není typu kontejner a nemůže obsahovat další prvky. Sama o sobě nemůže měnit data, avšak pomocí malých úprav lze docílit různého chování, dokonce i okamžité aktualizace záznamu.

Pokud je nutné ve formuláři přenášet více záznamů, je třeba použít komponentu Hidden, nastavit datový typ LN (ListOfNumbers) a na formuláři pomocí JavaScriptu zajistit naplnění dle potřeby.

4.6.9 TabControl

Je třeba podotknout, že TabControl není jen komponenta, ale má povahu řídicího prvku. Na formulář lze do kořene umístit jen komponentu typu Layout nebo TabControl, aby je framework správně zobrazil. Tato komponenta umožňuje členit formulář do skupin a dle zvolené zavolané akce aktivovat či skrýt záložky. Tím se umožňuje variabilita formuláře například pro požadované rozdílné chování při různých uživatelských rolích. Není potřeba vytvářet nové okno, případně programovat jinou logiku, stačí jen vypsat seznam záložek, které se mají pro danou akci na formuláři aktivovat. Ostatní záložky zůstanou opět nezobrazené. Pomocí záložek lze vytvářet průvodce.

4.6.10 Layout

Komponenta Layout je komponenta typu kontejner, která umožňuje návrháři pracovat s maximální možnou zobrazenou plochou, alespoň pokud je nastavena vlastnost scrollable. Pokud je komponenta zobrazena s posuvníky, předpokládá se použití na formuláři ve stylu hlavička/detail, kde detail má proměnlivou velikost. Aby uživatelé nemuseli vytvářet složité CSS stylování, je tato komponenta ovládána pomocí JavaScriptu. Uživatel si pouze zvolí chování. Předpokládá se použití pouze jednoho scrollable layoutu na formuláři. Výjimka je použití scrollable panelu uvnitř záložky v komponentě TabControl, kde je toto omezení přesunuto z formuláře na záložku. Pokud je nutné použít více posuvných komponent, je třeba použít komponenty typu panel a řízení pomocí CSS nebo HTML a chování prvků řídit pomocí JavaScriptu.

4.7 Akce

Framework je založen na principu akcí. Ty jsou vyvolávány z klientské strany a server na požadavky reaguje. Akce se skládají z několika částí. Než je vlastní akce spuštěna, je spuštěna sekvence kontrol, pokud jsou deklarované. Kontrola je založena na spuštění zdroje *ISimple. V*ýsledek je uložen do proměnných, které pak následně mohou být porovnávány. Kontroly jsou

koncipovány na principu selhání. V případě, že je podmínka kontroly splněna, znamená to, že vlastní akce nemůže být spuštěna a místo aktivního kódu definované akce se spustí kód uvedený v kontrole se splněnou podmínkou.

Ke spuštění vlastního kódu akce dojde pouze v případě, že u všech kontrol nebudou splněny podmínky, nebo pokud nejsou definovány žádné kontroly. Kód akce je realizován spuštěním metody ***javascript*** instance třídy implementující rozhraní *IActionExecutable*. Metodou je vrácen klientský kód, který spustí požadovanou akci

Kontroly lze použít i jiným způsobem – mohou vyvolávat dialogová okna, ve kterých lze podle splněné kontroly přesměrovat akci. Jako příklad by mohl posloužit tisk sestavy. Pokud jsou splněny všechny podmínky, lze spustit tisk. Pokud chybí položky, lze spustit dialogové okno, které uživatele upozorní, že něco není v pořádku a pokud souhlasí, spustit akci, kterou data může uvést do pořádku.

Tabulka 9 – Seznam podporovaných akcí

Název	**Popis**
ShowActionWindow	spuštění dialogového okna
downloadElDocument	stažení dokumentu ze serveru na klienta
exportAcutalFolder	export vybraného adresáře s vybraným filtrem a setříděním do CSV
exportSqlCommand	export vybraného zdroje do CSV
messageBox	zobrazení dialogu
runBatch	přímé spuštění dávky
sendEmail	odeslání e-mailu
uploadDocuments	odeslání jednoho či více souborů z klienta na server, kde jsou zpracovány jako byte array předanou aktualizační dávkou

Některé akce spuštěním svého kódu končí. V ostatních případech však akce pokračují spuštěním dávky. Dávkou se rozumí instance třídy implementující rozhraní *IBatch* z modulu zdrojů. Dávek může být více. Framework umožňuje

dávky řetězit. Pokud je dávka provedena bez chyby, je spuštěn kód určující způsob chování frameworku po vykonání kódu. Standardně jsou to akce:

- nedělat nic
- obnovit detail
- obnovit vše
- zobrazit záznam

Spolu s reakcí je umožněno spustit návaznou akci či spustit vlastní klientský skript.

4.8 Serverové moduly

Serverové moduly jsou vlastně aktivní kód serveru reagující na požadavky klienta. Moduly jsou vyvíjeny na bázi Generic Handler v asynchronním módu (**AsyncJSON.ashx**). Jsou tedy nejdůležitější součástí frameworku. Pokud by nebyly napsány serverové moduly, nebyl by framework schopen reagovat na požadavky klienta. Načítání serverových modulů je opět založeno na .NET reflexi. Aby byla akce zařazena do aktivního spouštění, je nutné v požadované třídě implementovat rozhraní *IRequestDecoder*. Po startu aplikace dojde pomocí reflexe k načtení tříd implementujících rozhraní.

Tabulka 10 – Popis definice rozhraní IRequestDecoder

Metoda/vlastnost	Výsledek	Druh	Popis
action	string[]	vlastnost	seznam jmen, pod kterými lze modul vyvolat
proces	string	metoda	provádí požadovanou akci; výsledkem operace je klientský kód
directToStream	bool	metoda	příznak, zdali je výsledek zpracováván jako JavaScript nebo jsou obsahem binární data
CreateObject	IRequestDecoder	metoda	slouží k Dependency Injection

Pro většinu registrovaných akcí platí, že zároveň s vlastním kódem akce existuje i serverový modul, který zpracovává data generovaná akcí. Není to však podmínkou. Serverové moduly nemusí sloužit pouze jako výkonný kód akce, ale mohou být spouštěny i samostatně a vykonávat operace nezávisle na frameworku na vyžádání klienta.

Tabulka 11 – Implementované serverové moduly

Název modulu	Účel
create_filter	vytvoření okna filtru pro vybranou složku
download_document	stažení binárního souboru – dokumentu
export_actual_folder	export do CSV souboru dle aktivní složky i s aktivními filtry a tříděním
export_sql_command	vlastní export dle vybraného zdroje
filter_combo_list	seznam hodnot pole filtru typu ListOfValues
filter_subquery	seznam hodnot pole filtru typu Subquery
get_folder_count	dotaz na počet záznamů složek přihlášeného uživatele ve vybrané roli; počítají se pouze záznamy složek, které jsou označeny atributem showCount
get_sql_list	vrácení dat dle vybraného definovaného zdroje
get_sql_row	vrácení JSON objektu ze zdroje; pouze první řádek
save_d.a.window	uložení záznamu z dynamického akčního okna (uživatelského formuláře)
send_email	odeslání e-mailu
run_action_batch	vyvolání spuštění dávky akce

Seznam metod, které framework umožňuje spustit, je vypsán při startu aplikace, pokud je zapnuté ladění. Seznam implementovaných akcí je shrnut v tabulce 11. Jde o kontrolu, že framework správně načetl všechna rozšíření.

5 SERVER–CLIENT rozhraní

V této kapitole budou popsány obecné principy komunikace server–client, které jsou využívány výslednou aplikací.

5.1 HTTP (S) komunikace

HTTP či zabezpečený HTTPS protokol je vůbec základní komunikační protokol vytvořený právě pro internet. Jeho principy a metody budou vysvětleny v následujících podkapitolách.

5.1.1 Princip

HTTP (Hyper Text Transport Protocol) je aplikační protokol nad transportním protokolem TCP. Jedná se o uzavřenou komunikaci založenou na principu otázka–odpověď (request–response). Dle verze protokolu HTTP 1.0 byla komunikace po odeslání odpovědi uzavřena a spojení přerušeno, což bylo neefektivní. Aby nedocházelo k uzavírání spojení, využívala se k tomuto účelu hlavička klienta *„Connection: Keep-Alive"*. Server pak věděl, že má očekávat další požadavky. Od verze HTTP 1.1 se všechna spojení považují za trvalá, pokud není uvedeno jinak. [16]

Standardně jsou pro HTTP komunikaci vyčleněny porty 80, 8080, 8008. Pro komunikaci je velmi často využíván i protokol HTTPS (Hyper Text Transport Protocol Secure), což je HTTP zabalený do SSL/TLS běžící standardně na TCP/443. Jedná se o šifrované spojení, aby nikdo nemohl odposlouchávat komunikaci a získávat přístupy k uživatelským účtům.

5.1.2 Metody HTTP

Metody HTTP jsou používány k identifikaci zdroje, který je požadován. Na základě rozpoznání metody lze identifikovat obsah a řídit odpověď. Dle RFC [17] jsou rozlišovány tyto základní metody komunikace:

- OPTIONS – metoda reprezentuje požadavek o dostupných možnostech a/nebo požadavcích spojených s prostředky, aniž by to znamenalo zahájení získávání zdrojů; odpověď nelze ukládat do mezipaměti.
- GET – je metoda znamenající získávání informací ze serveru. Je to nejpoužívanější metoda, odpověď lze ukládat v mezipaměti. Aby nedocházelo k vracení zastaralých informací, je tato metoda doplněna řadou metainformací, např. jak je dokument starý či kdy byl modifikován. Obvykle nemá tělo.
- HEAD – je stejná metoda jako GET, jen s tím rozdílem, že nevrací žádné tělo (obsah dokumentu). Je používána k získávání informací o dostupnosti dokumentu.
- POST – metoda používaná k odeslání většího množství dat. Data jsou odesílány v těle požadavku. Většinou se používá k odeslání souborů na server.
- PUT/DELETE – vytvoření či smazání objektu ze serveru. Metody jsou využívány v kombinaci s užitím techniky REST.
- TRACE – metoda je používána k trasování požadavku.
- CONNECT – metoda slouží k tunelování HTTP protokolu a je využívána k navazování spojení SSL.

5.2 AJAX

Komunikace mezi klientem a serverem je ve frameworku založeno na technologii AJAX (Asynchronous JavaScript and XML). Standardní ASP.NET je založen na odesílání celé webové stránky na server a zpět. Vybavení takového požadavku je časově nákladné a pomalé. Každá akce vyžadující interakci se serverem odešle obsah stránky na server, kde se obsah zpracuje, provede se vyvolaná akce a stránka se pošle zpět na klienta. AJAX je také používán k odesílání dat, ale neposílá se celý obsah webové stránky, pouze požadované informace. Celá akce probíhá (většinou) na pozadí. Webová stránka zůstává ve stejném stavu jako před odesláním. Aby mohla aplikace správně fungovat, je nutné mít na straně klienta povolený JavaScript.

Žádosti na serveru musí mít k dispozici správný nástroj, který bude umět požadavky z klienta zpracovat. V ASP.NET jsou pro tuto činnost ideální generic handlers nebo webservices.

5.3 Generic Handler

Generic handlers (ASHX) jsou dynamické soubory, jejichž obsah je plně generován programovým kódem. Není stanoveno, jakého druhu bude obsah. Např. u ASPX souborů (WebPages) se očekává, že obsahem je generovaný HTML kód. U generic handlerů je čistě na programátorovi, jaký obsah poskytne. Obsah se navíc může lišit dle poslaných parametrů i dle metody, kterou se handlery volají. Tato technologie je ideální pro tvorbu rozhraní client–server. Není vázána návratovým typem, obsah není předem definován, lze jej dynamicky upravovat a dle zvolené metody (služby) vytvořit odpovídající návratový kód. Generic handlery jsou vhodné i pro předávání binárního obsahu (PDF, doc, png apod.) a umožňují jak synchronní, tak asynchronní komunikaci.

5.4 WebService

WebService (ASMX) je standardní webová služba fungující dle standardu SOAP (Simple Object Access Protocol) [18]. Za normálních okolností SOAP komunikuje pomocí XML dokumentů. Ve vyšších verzích .NET lze nastavit chování webové služby tak, aby komunikovala ve formátu JSON. Tím se stává ideální na tvorbu komunikace s webovým klientem pomocí technologie AJAX. Rozdíl oproti generic handlers spočívá v pevně definovaném rozhraní. Zavoláním webové služby lze dostat seznam podporovaných metod a jejich parametrů a výsledek je vždy serializován do XML (JSON). Webovou službou nelze přenášet binární data v surové formě. Převodem na alternativní obsah narůstá objem komunikace.

5.5 WCFService

WCF (Windows Communication Foundation) je novější technologie od firmy Microsoft, která sdružuje starší komunikační protokoly a zjednodušuje práci s jejich rozhraním. V dřívějších verzích Visual Studia bylo možné najít několik

technologií, které umožňovaly komunikaci mezi aplikacemi – ASMX, .NET Remoting pro výměnu dat mezi aplikacemi, DCOM (Enterprise Services) nebo MSMQ (Message Queuing) [19].

WCF přináší funkcionalitu všech technologií dohromady pod sjednoceným programovým modelem, což zjednodušuje vyvíjení distribuovaných aplikací.

5.6 Komunikace Server–Client ve frameworku

Veškerá komunikace frameworku využívá standardního protokolu HTTP či HTTPS. Při využívání prostředků, jež nabízí samotný ASP.NET, jsou volány stránky ASPX. Jelikož jádro programu tvoří celkem 3 stránky, které zajišťují přihlášení, přehled a tvorbu dynamických oken, jedná se o komunikaci minimální. Veškerá ostatní komunikace již probíhá na principu AJAX, na server jsou odesílány požadavky s informacemi a výsledek zpracování probíhá na pozadí v prohlížeči na klientovi. Požadavky mohou být odeslány jako formulář nebo jako data ve formátu JSON.

Server zpracovává požadavky pomocí dvou generic handlerů, a to AjaxJSON.ashx a AsyncJSON.ashx. Zatímco AjaxJSON obstarává základní komunikaci server–client, a to synchronně, AsyncJSON obstarává volání serverových modulů asynchronně.

V některých případech jsou na server přijímány i požadavky pomocí metody GET. Ve většině případů se však jedná o požadavky očekávající binární data v odpovědi. Tyto požadavky jsou pak volány klientem přímo, nikoliv technologií AJAX.

6 Popis konfiguračních souborů

Framework využívá pro svůj chod několik konfiguračních XML souborů. Jednotlivé soubory souvisí s částmi architektury frameworku, a to konkrétně:

- resources.xml – seznam datových zdrojů
- treview.xml – definice uživatelských rolí, autentizace a nastavení frameworku
- buttons.xml – seznam dynamických tlačítek
- actions.xml – seznam deklarovaných akcí
- windows.xml – seznam uživatelských oken

Pro uvedená nastavení existuje i patřičné XSD, které umožňuje uživatelům lépe se orientovat ve struktuře nastavení a kontroluje korektnost vyplněného konfiguračního souboru.

Modernější editory jsou schopny s těmito definicemi pracovat a uživatelům nabízet nápovědu v podobě našeptávání. Není proto nutné si pamatovat jednotlivé atributy či elementy. Ze sestavování konfiguračních souborů se tak stává interaktivní vyplňování, kdy se v reálném čase navíc provádí i kontrola očekávaných hodnot. K nástrojům tohoto druhu patří XMLNotepad od firmy Microsoft nebo Visual Studio od stejného tvůrce.

6.1 Resources.xml

Recources.xml je konfigurační soubor korespondující s částí *zdroje* v architektuře frameworku. Sestavení konfiguračního souboru Resources koresponduje již s konkrétními třídami implementujícími rozhraní popsané v kapitole

Datové zdroje.

Tabulka 12 – Přehled základních tříd výběrů

Třída výběru	Popis
simple	jednoduchý dotaz do databáze; očekává se pouze jeden návratový řádek; počet sloupců není omezen, ale očekává se jednoznačné jméno
search	používá se pro komponenty s nabídkou; je rozdělen na dvě části, přičemž jedna část slouží jako zdroj dat pro nefiltrované dotazy a druhá samozřejmě pro dotazy filtrované
structure	strukturovaný výběr; používá se pro zobrazení dat v tabulkách; struktura výběru se rozpadá na komponenty SELECT, FROM, WHERE a ORDER; aby bylo možné řídit zobrazení, obsahuje i složku COLUMNS, která určuje pořadí a název sloupců zobrazených v tabulce

Každá třída má specifické využití. V základní verzi frameworku je vytvořena komunikace nad relačními databázemi typu Oracle, MSSQL a PostgreSQL. Přesto se jedná o třídy vycházející z rozhraní IDatabase. Lze tedy vytvořit vlastní třídy, které budou pracovat s jinými zdroji dat. Lze například pracovat s daty z externích REST API, webových služeb na bázi SOAPu nebo XML a XLS souborů.

Druhá část zdrojů, tedy operace frameworku (data processing), obsahuje veškeré možnosti nakládání s uživatelskými daty. Tabulka 13 popisuje třídy obsažené v základní verzi frameworku.

Tabulka 13 – Přehled operací pro zpracování dat

Operace	Popis
batch	základní rozhraní pro dávkové operace; jedná se o obálku podporovaných akcí
action	vlastní akce dávky – běžně se používá pro UPDATE, INSERT atd.

select	výběr v těle dávky je určen pro načtení dodatečných informací; očekává se pouze jeden řádek; sloupce z výběru jsou zpracovány jako proměnné a přidány k formulářovým hodnotám
email	v rámci dávky odesílá e-mail; používá se na odesílání informací uživateli, například potvrzení registrace, informace o stavu akce
failure	umělé vyvolání chyby pro případ, že nebyly naplněny některé podmínky splnění akce; provede se vyhodnocení podmínky, a pokud je podmínka splněna, provede se ukončení operace; v případě, že jsou operace v transakci, provede se rollback
retype	přetypování proměnné pro zpracování dávky
hash	vytvoření HASH byte array, který je překódován do hexadecimálního kódu převedeného na text

Jedná se o nejčastěji používané operace pro zpracování informací. Z větší části jimi lze pokrýt většinu operací požadovaných při zpracování dat. Některé operace chybí, například operace se soubory. Jsou to operace, u kterých nelze jednoznačně určit vhodné zpracování – někdo může soubory evidovat v databázi, někdo jiný může chtít ukládat soubory na discích, v cloudu apod. Pro tyto účely bude nutné napsat vlastní třídu implementující *IBatchAction*, která danou operaci bude schopna provést a modifikovat XSD, aby byl framework schopen novou operaci načíst.

Zdroje pracují s SQL příkazy a občas je nutné data omezit podmínkou, například u zdrojů pro detaily. Každá přehledová tabulka určuje tzv. pole typu system_columns, jež slouží jako identifikátory daného přehledu. Pokud framework potřebuje informace ze serveru, vybrané hodnoty s názvem sloupců odesílá spolu s požadavkem. Framework následně tyto informace zpracuje a vytvoří seznam dostupných proměnných. Na tyto proměnné se pak lze odkazovat ve zdrojích. Pro databázové servery je používána syntaxe „:oracle_param“, „@mssql_param“ apod.

Pokud jsou ve zdroji proměnné s výše uvedenou syntaxí, framework je vyhodnotí jako databázové proměnné a spolu s požadavkem na obsah odešle seznam požadovaných proměnných. Tento způsob je odolný vůči útoku SQL injection.

Pro účely frameworku je však možné použít i syntaxi „{param}“. Pokud je nalezena takováto proměnná, je brán v úvahu typ proměnné a předtím než se dostane SQL příkaz k vlastnímu zpracování, je proměnná nahrazena hodnotou. Tento způsob použití NENÍ odolný vůči SQL injection, existují však situace, kdy nelze s proměnnou pracovat jiným způsobem – jedná se převážně o proměnné typu pole.

6.2 Treeview.xml

Konfigurační soubor treeview.xml neobsahuje jen stromovou strukturu uživatelských rolí. Týká se i nastavení autentizace a některých stěžejních prvků frameworku. Vlastní struktura konfiguračního souboru je rozdělena do sekcí:

- nastavení
- autentizace
- systémové sloupce
- přehledy
- role

6.2.1 Nastavení

Nastavení je formou atributů kořenového elementu. Jedná se o doplňková nastavení. Framework bude bez tohoto nastavení fungovat také správně, pouze s výchozími hodnotami určenými vývojářem.

6.2.2 Autentizace

U autentizace lze zvolit metodu, kterou framework použije k autentizaci. Jsou na výběr dvě možnosti:

- databázová autentizace

- autentizace vůči SQL zdroji

Obě možnosti fungují téměř stejně. Jediným rozdílem je uživatel, který se hlásí k datovému úložišti. Pokud je zvolena databázová autentizace, je uživatel autentizován přímo databází svými přihlašovacími údaji. Vztahují se na něj tudíž i patřičná omezení v DBS. Pokud je zvolena autentizace vůči SQL zdroji, připojuje se aplikace pomocí vlastních údajů uložených v nastavení webové aplikace. Uživatelské přihlašovací údaje se ověřují vůči datovému zdroji (např. vůči tabulce v databázi) se seznamem uživatelů. Na přihlášeného uživatele se vztahují databázová práva uživatele uloženého v nastavení webové aplikace.

Aby bylo možné zvolit např. význam sloupců tabulky souvisejících s autentizací, jsou pro framework důležité následující informace:

- SQL zdroj pro ověření uživatele
- název sloupce s uživatelským jménem
- název sloupce s uživatelskou rolí
- metoda hash pro kontrolu hesla uloženého v databázi

Pomocí SQL zdroje je možné získat globální nastavení, které se uloží do autorizační cookie. Seznam sloupců je definován podřízenými elementy *column*, které slouží k provázání informací ze zdroje dat pro framework, jako jsou název proměnné a datový typ. Na proměnné se pak lze odvolávat ve zdroji pro účely filtrování záznamů.

6.2.3 Systémové sloupce

Systémové sloupce slouží frameworku k určení zájmových identifikátorů. Při komunikaci mezi serverem a klientem dochází k předávání proměnných. Server je však schopen doplnit pouze proměnné, které jsou v danou chvíli frameworku známé. Tímto výčtem framework rozšiřuje seznam známých parametrů společně s jejich typem. Díky tomuto principu lze provádět kontroly zobrazení tlačítek a záložek.

6.2.4 Přehledy (table)

Tabulka ve frameworku se skládá ze dvou HTML tabulek, z nichž jedna je použita na záhlaví a druhá zobrazuje vlastní data. Je nutné obě tabulky synchronizovat, tedy vypočítat šíři sloupců, kontrolovat horizontální posun, aby datovým sloupcům odpovídaly sloupce uvedené v záhlaví. Záhlaví zůstává ve frameworku neměnné, rolují pouze data.

Standardní HTML tabulka neumí rolovat jen obsah – posouvá se celá. Pomocí CSS3 lze sice docílit rolování obsahu, je to však dosažitelné jen pro některé prohlížeče.

Sekce table slouží k nastavení parametrů zobrazení tabulky. Tabulky jsou vytvářeny na serveru a na klienta jsou posílány jako HTML fragment. Při pokusech o vytváření tabulky přímo na klientovi nebylo dosaženo stejného výkonu jako při vytváření tabulek serverem.

6.2.5 Role

Uživatelské role jsou stavebním kamenem frameworku. Zde je k dispozici seznam uživatelských rolí dostupných uživatelům spolu s jejich skladbou. Pro každou roli je definován seznam dostupných detailů (detailních záložek), seznam uživatelských tlačítek (statická tlačítka vpravo nahoře) a hlavně seznam dostupných složek agendy.

Pro detaily se zde definuje, jaká okna se pro daný detail zobrazí a s jakým SQL zdrojem bude okno iniciováno. Tím je zaručeno, že pro danou roli jsou iniciovány správné SQL dotazy a detaily zobrazují správné informace zvolené rolí.

Seznam dostupných složek je členěn do skupin pomocí stromové struktury. Složky mohou mít přímou vazbu na přehledový strukturovaný SQL dotaz (třída s implementovaným rozhraním *IStructured*) nebo mohou mít vazbu na okno, čímž je zajištěno, že nebude zobrazena standardní obrazovka, ale uživatelsky definovaný pohled na část dat, např. DASHBOARD. Pokud není nastavena ani jedna

z vlastností, po výběru takové složky se zobrazí její popis a popis podřízených složek. Pro případ, že byla zvolena možnost přehledové tabulky, je možné nadefinovat kontextové menu, které umožňuje nad vybraným záznamem spouštět dodatečné akce.

Tabulka 14 – Atributy pro nastavení složky

Atribut	Popis
Title	titulek složky; popisuje obsah složky
Comment	doplňující informace rozšiřující základní fakta o složce; komentář je zobrazen nadřazenou složkou, pokud u ní není vyplněn SQL dotaz nebo vazba na okno
Css	nastavení zobrazení složky ve stromové struktuře; umožňuje přidělit složce ikonu
Default	pokud je složka označena jako default, je při načtení stránky zobrazen obsah této složky
showCount	pokud je tento atribut nastaven, provádí se v pravidelných časových intervalech počítání záznamů ve složkách; tímto atributem lze počítání potlačit nebo naopak zapnout
Bold	nastavuje zvýraznění složky
allowPaging	povoluje stránkování záznamů v přehledové tabulce; ve výchozím stavu je stránkování zakázáno
Id	identifikátor složky; měl by být jednoznačný
empty	tento atribut se používá v případě, že je potřeba opticky oddělit některé skupiny složek, složka je pak vykreslena jako prázdné místo

6.3 Actions.xml

Actions.xml slouží k deklaraci všech dostupných akcí projektu. Jedná se o konfigurační dokument sestavující dohromady zobrazení uživatelských dialogů,

SQL zdrojů, aktualizace dat a následné akce po dokončení požadované operace. Před provedením akce lze spouštět kontroly umožňující dodatečné reakce systému. Těmito kontrolami lze zamezit neoprávněnému spouštění akcí nebo např. spouštět rozhodovací dialogy.

Akce ve frameworku nemusí vždy znamenat spuštění okna (uživatelského dialogu), ale umožňuje např. odeslat e-mail, zobrazit zprávu nebo spustit aktualizační dávku.

Vlastní sestavení akce se skládá z několika částí:

- nastavení dočasných proměnných
- kontrola
- spuštění
- aktualizace
- ukončení

6.3.1 Nastavení dočasných proměnných

Nastavení dočasných proměnných slouží frameworku k ošetření neočekávaných situací. Místo aby se uživatel snažil dostat proměnné do zdrojových dotazů, může nastavit výchozí hodnoty před akcí.

Proměnným je nastaveno jméno, typ a výchozí hodnota. Pokud se během zpracování uživatelských dialogů vyskytne proměnná se stejným názvem dočasné proměnné, je výchozí hodnota této dočasné proměnné přepsána hodnotou z uživatelského dialogu.

6.3.2 Kontrola

Kontrola před spuštěním akcí nemusí být uvedena. Pokud je uvedeno více kontrol, jsou postupně spouštěny v pořadí, v jakém jsou definovány.

Kontrola je svázána s SQL zdroji. Je proveden dotaz s parametry, které jsou předány akcí a které mohou záviset na vybraném záznamu, uživatelském profilu

nebo např. na zvolené složce. Poté, co je proveden dotaz, je postupně vyhodnocován stav. Případné splnění podmínky je v případě kontrol u akcí chápáno jako výjimečný stav, na který je nutné reagovat.

Reakcí systému se rozumí např. zobrazení zprávy, dialogu, opětovné načtení přehledu či detailu, nebo nemusí být provedeno vůbec nic.

6.3.3 Spuštění

Spuštění slouží k určení, jakým způsobem se má provést výkonný kód. V nejčastějších případech je využíváno spouštění uživatelských dialogových oken (formulářů), které slouží k aktualizaci či vytváření nových záznamů.

Tabulka 15 – Seznam podporovaných akcí

Akce	Účel
exportActualFolder	export aktuálně vybrané složky spolu s nastaveným filtrováním a tříděním; data jsou exportována do formátu CSV; ostatní parametry exportu, jako je kódování či oddělovač, jsou nastavitelné parametry akce
ExportSqlCommand	stejné jako předchozí akce, jen s tím rozdílem, že nereaguje na vybranou složku, ale na uživatelsky zvolený SQL zdroj.
messageBox	zobrazí dialogové okno zprávy; zpráva může mít charakter informační, varování, chyby nebo dotazu; stejně jako lze zvolit charakter dialogu, lze nastavit tlačítka, která budou na dialogu zobrazeny
runBatch	přímé spuštění aktualizační dávky; pokud akce nepotřebuje žádnou interakci od uživatele, lze přímo spustit aktualizační dávku; toto se využívá při řetězení akcí
sendEmail	odeslání e-mailu
uploadDocuments	upload dokumentu (dokumentů); spustí dialog, který umožní odeslat soubory z klienta na server, kde jsou

	data uložena do vybraného úložiště (např. databáze, datový sklad či adresářová struktura)
downloadDocument	stažení uloženého souboru z úložiště na klienta

6.3.4 Aktualizace

Sekce aktualizace dat slouží k sestavení aktualizačních dávek pro vytváření, aktualizování či rušení záznamů. Jednotlivé dávky lze řetězit. Není nutné spouštět jen jednu dávku. Dávky lze řetězit uvedením názvů za sebe oddělených čárkou, nebo uvedením pod sebe v XML. Rozdíl je v tom, že pokud jsou uvedeny za sebou oddělené čárkami, lze provedené změny uzavřít v transakci. Pokud jsou uvedeny pod sebou, transakční bloky se na ně nevztahují.

Vyjmenování dávek má vztah ke zdroji –je očekáván název zdroje implementujícího rozhraní *IBatch*. Pokud tomu tak není, dojde k vyvolání chyby a ukončení skriptu chybou. V případě vytvořené transakce se provede Rollback.

6.3.5 Ukončení

Ve chvíli, kdy dojde k aktualizaci dat, je vlastně celá akce téměř u konce. Zapsáním změn do datového úložiště byla úspěšně dokončena celá akce. V tomto případě dochází ke změně záznamů, které mohou být v danou chvíli zobrazeny na klientovi. Pro tyto případy je nutné ještě nastavit, co se má po dané akci provést – zda stačí jen provést aktualizaci detailu, nebo je třeba provést kompletní znovunačtení stránky a přepočítat záznamy. Také je možné ukončit akci bez jakékoli akce. Např. odesláním e-mailu se nemusí měnit data v přehledech, a proto není třeba provádět žádné akce.

Další možností je potřeba spuštění nějaké jiné akce, popř. spuštění uživatelského skriptu.

Tabulka 16 – Seznam ukončovacích akcí

Akce	**Účel**

noRefreshAction	neprovede žádnou akci na klientovi, pokud nebude svázána s akcí či uživatelských skriptem
refreshAll	kompletní načtení přehledu a přepočítání záznamů ve složkách; přepočítávání záznamů lze potlačit
refreshTab	znovunačtení vybraného detailu
showObject	zobrazení záznamu v přehledu; tato akce je používána pro zobrazení vytvořeného záznamu a umožňuje spuštění řetězové akce nad čerstvě vybraným záznamem

6.4 Windows. xml

Posledním z konfiguračních souborů je windows.xml. Jedná se o seznam uživatelských dialogů (formulářů), které se v rámci aplikace spouští. Dialogová okna se zobrazují jak při zobrazovaní detailů záznamů, tak při tvorbě či aktualizaci záznamů. Obsahem se podobá HTML.

Dialogová okna mohou být spustitelná jak v závislosti na přihlášení, tak bez autentizace. Mohou nastat případy, kdy není potřeba být přihlášen, a přesto je třeba vyplnit formulář. Jedním z takových případů je registrace nového uživatele.

Dialogová okna jsou skládána pomocí komponent, které jsou ve frameworku registrovány (viz kapitola 4.6). Nemusí to však být konečný výčet komponent. Počet komponent se dle potřeby může rozšiřovat, aby bylo možné vyhovět zadání, které nelze realizovat za použití implementovaných komponent.

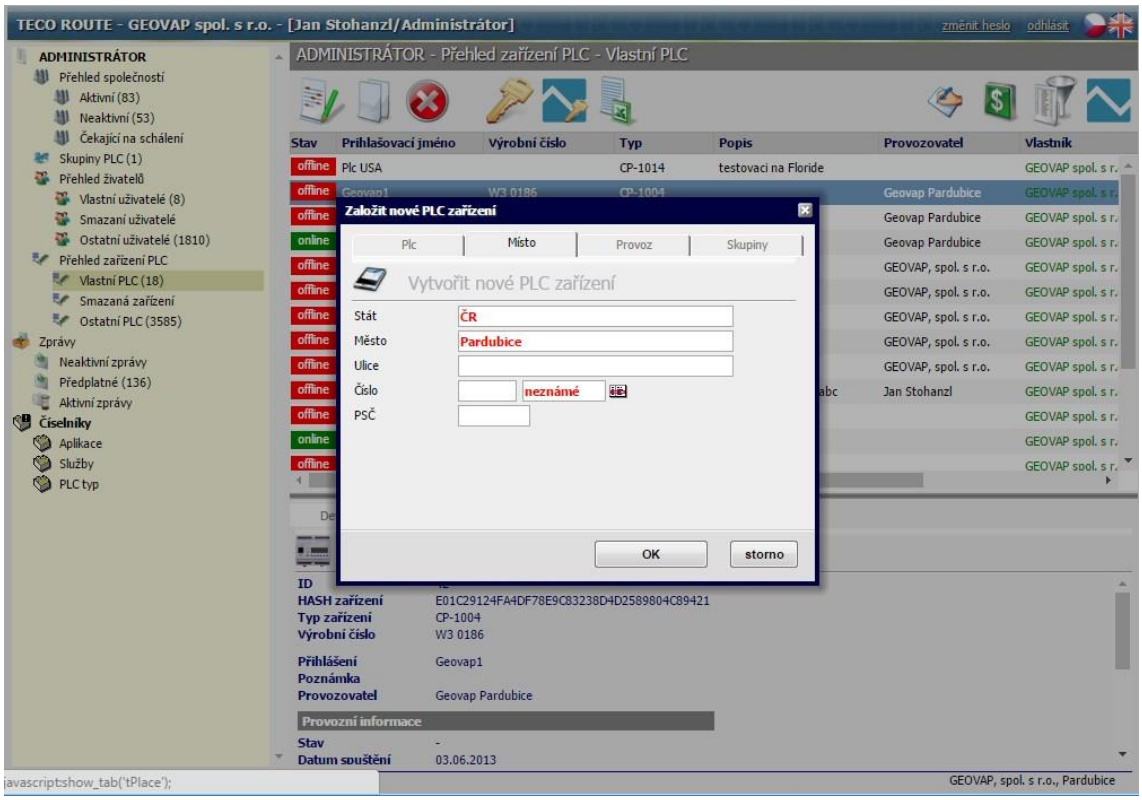
Tabulka 17 – Seznam elementů pro windows.xml

Element	Popis
defaultValues	stejně jako u akcí umožňuje zaregistrovat dočasné výchozí hodnoty vztažené k uživatelskému dialogu
checkBeforeOK	umožňuje před vlastním provedením změn zkontrolovat hodnoty, např. pro zamezení nežádoucích změn či neoprávněného zásahu do dat
script	umožňuje vepsat do uživatelského dialogu vlastní

	JavaScriptový kód bez nutnosti zapisovat tuto část kódu do k tomuto účelu sloužícímu souboru user_defined_function.js; odpadají problémy s CACHE browseru
tabControl, layout	základní stavební prvek uživatelských dialogů
css	uživatelské styly nad formulářem; stejně jako script umožňují uživateli dynamicky přizpůsobovat CSS styly na uživatelském dialogu bez nutnosti zápisu do přiložených css souborů; nehrozí problémy s CACHE

7 Shrnutí výsledků

Závěrečná práce je zaměřena na Deklarativní Framework, jehož cílem má být tvorba webových aplikací typu agenda, které se mají chovat jako desktopová aplikace. Při průzkumu existujících nástrojů bylo zjištěno, že se vyskytují technologie, které by mohly tomuto účelu sloužit a nástroje, které se nejvíce blížily požadavkům zadání práce, byly zmíněny ve 2. kapitole. Zde byly navíc popsány vývojářské technologie, které jsou v dnešní době z pohledu autora nejzajímavější pro tvorbu vlastní aplikace. Pro účely diplomové práce byla vytvořena ukázková konfigurace nad databází Microsoft SQL. Aplikace byla vytvořena tak, aby bylo možné prokázat funkčnost konceptu. Ukázková konfigurace obsahuje dvě tabulky a je napsaná pro tři uživatelské role. Lze pomocí ní vytvářet záznamy, modifikovat stávající záznamy, nebo mazat.



Obrázek 13 - praktická ukázka reálného použití

Pomocí frameworku byly vytvořeny testovací webové aplikace. Jedna z konfigurací je zobrazena na obrázku 13. Obsahuje 4 uživatelské role, bylo použito

okolo padesáti uživatelských dialogů, definováno je přes 220 SQL zdrojů a více než 80 akcí. Data jsou uložena v databázi PostgreSQL, největší z tabulek agendy je tabulka PLC zařízení se 4 tisíci záznamů. Aplikace běží v režimu 24/7 a denně se k ní připojují tisíce uživatelů.

8 Závěry a doporučení

Diplomová práce se zabývala vývojem aplikací, které by bylo možné konfigurovat pomocí XML. V rámci diplomové práce vznikl Framework, který je konfigurovatelný pomocí XML. Po změně konfigurace není třeba provádět dodatečné kompilace, aplikace reaguje na změny okamžitě. Z tohoto pohledu bylo zadání diplomové práce naplněno.

V případě změny atributu nad tabulkami datového modelu výsledné aplikace, nedojde k selhání celé aplikace, max. dojde k zobrazení chybového hlášení v případě, že dojde k aktivaci chybového kódu obsluhující kritické místo.

Aplikace je v tuto chvíli koncipována pouze pro použití v desktopových prohlížečích. Při použití mobilního zařízení nejsou využity obslužné procedury, které by umožňovaly plnou obsluhu, nicméně aplikace je zobrazitelná a plně funkční, obzvláště na zařízeních s vyšším grafickým rozlišením. Z hlediska dalšího vývoje aplikace by bylo vhodné aplikaci upravit, tak aby byla schopná pracovat plnohodnotně i na mobilních zařízeních. Tím by byla pokryta vyšší dostupnost výsledných aplikací.

Další návrh pro tvorbu by mohl stejně tak být směrován na tvorbu plnohodnotného mobilního klienta, bez použití prohlížeče. Tzn. stávající aplikační rozhraní rozšířit např. o REST rozhraní, případně webové služby, která by byla schopna předat konfiguraci a připravit pro uživatele nad danou platformou požadované prostředí.

Konfigurační soubory jsou ve své podstatě textové soubory, což by mohlo být bráno jako překážka při tvorbě nových konfigurací, proto logickým krokem k rozšíření by bylo vytvoření WYSISYG (What You See Is What You Get) editoru, který by usnadnil konfiguraci a následnou publikaci výsledné aplikace.

9 Seznam použité literatury

[1] *M. Patočka, „Informační systém Masarykovy univerzity," 2008. [Online]. Available: http://is.muni.cz/th/60871/fi_m/thesis.pdf. [Přístup získán 27 Březen 2016].*

[2] *„TIOBE index," TIOBE Software BV, [Online]. Available: http://www.tiobe.com/tiobe_index?page=index. [Přístup získán 27 Březen 2016].*

[3] *L. Srb, „Potřebné jemnosti u displejů jsme dosáhli, teď prosím čitelnost," Mobilizujeme.cz, 05 Květen 2013. [Online]. Available: http://mobilizujeme.cz/clanky/potrebne-jemnosti-u-displeju-jsme-dosahli-ted-prosim-citelnost/. [Přístup získán 26 Březen 2016].*

[4] *DevExpress, „Application Model," 25 únor 2016. [Online]. Available: https://documentation.devexpress.com/#eXpressAppFramework/CustomDocument112579.*

[5] *„Representational State Transfer," WikipediE, 09 Červenec 2015. [Online]. Available: https://cs.wikipedia.org/wiki/Representational_State_Transfer. [Přístup získán 28 Březen 2016].*

[6] *M. Malý, „REST: architektura pro webové API," 7 srpen 2009. [Online]. Available: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.*

[7] *N. C. Zakas, J. McPeak a J. Fawcett, Ajax Profesionálně, Brno: Zoner Press, 2007.*

[8] *M. Michálek, „K čemu je dobrý Bootstrap a frontend frameworky?," Zdrojak.cz, 06 prosinec 2013. [Online]. Available: https://www.zdrojak.cz/clanky/k-cemu-je-dobry-bootstrap-frontend-frameworky/. [Přístup získán 03 03 2016].*

[9] *„Bootstrap," Twitter, 19 duben 2011. [Online]. Available: http://getbootstrap.com/. [Přístup získán 03 březen 2016].*

[10] *J. Kosek, XML pro každého, Grada Publishing, 2000.*

[11] *J. Kosek ml., „DTD -- Definice typu dokumentu pod lupou," 27 únor 2016. [Online]. Available: http://www.kosek.cz/clanky/xml/xml-01.html.*

[12] *P. Bříza, „Základy jazyka XPath," 9 duben 2004. [Online]. Available: https://www.interval.cz/clanky/zaklady-jazyka-xpath/.*

[13] *J. Kosek, „Jazyk XSL," 27 únor 2016. [Online]. Available: http://www.kosek.cz/clanky/swn-xml/xsl.html.*

[14] *J. Clark, „XSL Transformations (XSLT)," 16 listopad 1999. [Online]. Available: https://www.w3.org/TR/xslt.*

[15] *„ASP.NET Application Life Cycle Overview for IIS 7.0," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/bb470252(v=vs.100).aspx. [Přístup získán 28 Březen 2016].*

[16] F. Kučera, „Protokol HTTP,“ 12 09 2011. [Online]. Available: https://www.zdrojak.cz/clanky/protokol-http/.

[17] „Hypertext Transfer Protocol -- HTTP/1.1,“ 1 červen 1999. [Online]. Available: https://tools.ietf.org/html/rfc2616#section-5.1.1.

[18] „SOAP Specifications,“ W3C, [Online]. Available: https://www.w3.org/TR/soap/. [Přístup získán 28 Březen 2016].

[19] „Služby Windows Communication Foundation a služby WCF Data Services v sadě Visual Studio,“ Microsoft, [Online]. Available: https://msdn.microsoft.com/cs-cz/library/bb907578.aspx. [Přístup získán 28 Březen 2016].

[20] B. Bernard, „Úvod do architektury MVC,“ 7 květen 2009. [Online]. Available: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/.

[21] „Can I Use,“ [Online]. Available: http://caniuse.com/#feat=input-datetime. [Přístup získán 27 Březen 2016].

10 Přílohy

1) CD se archívem zdrojového kódu a testovací konfigurací; zdrojový kód obsahuje složky:
   - Database – projekt obsahující komponenty pro připojení k databázím
   - GPFramework.Authentication – projekt starající se o autentizaci a autorizaci frameworku
   - GPFramework.Components – projekt obsahující serverové moduly, akce, dynamická tlačítka a komponenty oken
   - GPFramework.Resources – projekt starající se o zpracování SQL zdrojů
   - GPInterfaces – projekt ve kterém jsou definována rozhraní frameworku
   - ParseSQL – projekt starající se o parsování SQL příkazu
   - Portal – webová aplikace frameworku
   - SETTINGS – ukázková konfigurace frameworku s create skriptem datového modelu aplikace
   - WebControl – projekt starající se o zpracování složek

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

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

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Stohanzl Jan Dipl.tech.	Jana Zajíce 953, Pardubice - Studánka	I14300

TÉMA ČESKY:

Framework pro deklarativní vývoj aplikací pomocí XML

TÉMA ANGLICKY:

XML-based Framework for Declarative Application Development

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Framework pro deklarativní vývoj aplikací pomocí XML by měl sloužit pro rychlé a jednoduché vytvoření webové Single Page Application. Formou konfiguračních XML by měl být uživatel schopen vytvořit aplikace bez znalostí programovacího jazyka, či podpory programátora. Aplikace jako taková by měla být řízena pomocí deklarovaných událostí, které se budou lišit podle uživatelských práv a nastavených pravidel.

Obsah:
1. ÚVOD
2. Exitující nástroje
3. ANALÝZA
4. Architektura frameworku
5. SERVER-CLIENT rozhraní
6. Popis konfiguračních souborů
7. Shrnutí výsledků
8. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

ZAKAS, Nicholas C. JavaScript pro webové vývojáře. Vyd. 1. Brno: Computer Press, 2009, 832 s. Programujeme profesionálně. ISBN 978-80-251-2509-0
ZAKAS, Nicholas C, Jeremy MCPEAK a Joe FAWCETT. Ajax profesionálně. Brno: Zoner Press, 2007, 668 s. Encyklopedie webdesignera. ISBN 978-80-86815-77-0.
ENZ, Brian a John R DURANT. XML programming bible. New York: Wiley, 2003, xxxviii, 945 p. ISBN 07-645-3829-2.

Podpis studenta: ..... **Datum:** 22.4.2016

Podpis vedoucího práce: ..... **Datum:** 22.4.16