Systém pro správu a automizaci vydávání změn do Salesforce CRM

Bc. Jan Luksch

Diplomová práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Luksch**
Osobní číslo:	**A15368**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Systém pro správu a automatizaci vydávání změn do Salesforce CRM**
Téma anglicky:	**A System for the Administration and Automatic Release of Changes to the CRM Salesforce**

Zásady pro vypracování:

1. **Proveďte literární rešerši současných řešení pro vydávání změn pro platformu Salesforce CRM.**
2. **Analyzujte možná řešení a navrhněte systém pro správu vydávání změn do Salesforce CRM.**
3. **Vyhodnoťte bezpečnost procesů a realizujte zvolené řešení.**
4. **Vyhodnoťte výstupy projektu a proveďte jeho kritickou diskuzi.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.**
2. **KANISOVÁ, Hana a Miroslav MÜLLER. UML srozumitelně. Brno: Computer Press, 2004. ISBN 80-251-0231.**
3. **KRAVAL, Ilja. Analytické modelování informačních systémů pomocí UML v praxi, 1. vyd. Lipina : Object Consulting, 2010. ISBN 978-80-254-6986-6.**
4. **SOMMERVILLE, Ian. Softwarové inženýrství. Brno: Computer Press, 2013, 680 s. ISBN 9788025138267.**
5. **An Introduction to Environments, 2016. In: Salesforce Developers [online]. San Francisco. Dostupné z: https://developer.salesforce.com/page/An_Introduction_to_Environments.**
6. **Salesforce Security Guide [online], 2016. Version 38.0, Winter '17. San Francisco. Dostupné z: https://resources.docs.salesforce.com/204/latest/en-us/sfdc/pdf/salesforce_security_impl_guide.pdf.**

Vedoucí diplomové práce: **prof. Mgr. Roman Jašek, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **1. prosince 2017**

Termín odevzdání diplomové práce: **16. května 2018**

Ve Zlíně dne 11. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 9.5.2018 .....

podpis autora

ABSTRAKT

Tato práce se zabývá analýzou a implementací systému pro vydávání změn do Salesforce CRM. V rámci teoretické části jsou popsány funkce, výhody a nevýhody současných řešení. Dále text přechází do rozboru použitých technologií ve vlastním řešení, které reprezentuje webová aplikace. Analytická část práce popisuje požadavky na systém a přináší detailní rozbor případů užití včetně scénářů interakcí mezi uživatelem a systémem. Součástí je i návrh doménového modelu. V rámci popisu implementace jsou rozebrány důležité funkce systému včetně diagramů souvisejících tříd. Nedílnou součástí práce je i analýza a implementace bezpečnostních opatření proti hrozbám webových aplikací.

Klíčová slova: Salesforce, vydávání změn, migrace metadat, migrace dat, Apex skripty, ASP.NET Core, Entity Framework, ASP.NET Core Identity, PostgreSQL

ABSTRACT

Thesis deals with analysis and implementation of system which is dedicated for administration and automatic release of changes to Salesforce CRM. In theoretical part, there is an analysis of current solutions which includes brief description and list of benefits and disadvantages. Following part describes technologies used in own system which is represented by web application. Analytic part of thesis brings description of functional and non-functional requirements together with detailed use case analysis and scenarios of interactions between user and the system. Implementation part of thesis is focused on description of core functionalities including diagrams of related classes. Text is also concerned about detailed analysis of vulnerabilities which relate to web application development.

Keywords: Salesforce, version release, change management, metadata migration, data migration, Apex script run, ASP.NET Core, Entity Framework, ASP.NET Core Identity, PostgreSQL

Touto cestou bych rád poděkoval panu prof. Mgr. Romanovi Jaškovi, Ph.D. za cenné rady při vedení diplomové práce.

Dále bych rád poděkoval své přítelkyni za podporu, pochopení a trpělivost, kterou vynaložila během vzniku této práce.

OBSAH

ÚVOD

Salesforce.com je americká společnost zabývající se vývojem a poskytováním business systémů. Nejvíce se proslavila svým stejnojmenným systémem pro řízení vztahů se zákazníky (CRM), kterým si kladou za cíl usnadnit běžné činnosti firem, jako např. prodej, nákup, sklad, marketing, péče o zákazníky i kolaborace s obchodními partnery. Systém je poskytován jako cloudová služba, kterou si zákazníci předplácí. Společnost Salesforce.com zastává přístup zvaný software jako služba (Software as a Service - SaaS) (Jeff Douglas, 2010).

Jelikož není možné pokrýt veškeré požadavky firem, nabízí Salesforce mnoho možností, jak si systém přizpůsobit. Databáze systému lze rozšířit o vlastní struktury a funkcionality systému mohou být obohaceny pomocí Force.com platformy. Ta disponuje programovacím jazykem Apex pro řízení vnitřní logiky a technologiemi Visualforce či Lightning pro vývoj vizuálních částí aplikací. Zároveň je možné vytvářet vlastní webové aplikace, které mají výhodu přímého přístupu do databáze systému. Komunita kolem Force.com platformy se často rozděluje na administrátory, kteří se soustředí na konfigurace systému, a na vývojáře, kteří rozšiřují funkcionality systému pomocí vlastního kódu.

Při vývoji nové funkcionality je běžné, že implementace neprobíhá přímo na produkčním prostředí, na kterém pracují běžní uživatelé s reálnými daty. Chyby vzniklé během vývoje by mohly vést k odstávce systému, ztrátě dat a dalším negativním důsledkům. Systém Salesforce nabízí vytvoření tzv. sandboxů, které představují nezávislé kopie produkčního prostředí. Sandbox obsahuje stejné konfigurace a metadata, které v produkčním prostředí existovaly v momentu začátku klonování. Za příplatek společnosti Salesforce je možné přesunout i záznamy z databáze (An Introduction to Environments, 2016). Sandbox slouží jako vývojové prostředí, kde programátor může pracovat bez rizika, že by ohrozil provoz na produkčním prostředí. Salesforce nabízí možnost vytvořit hned několik sandboxů a je zvykem mít více prostředí pro vývoj a různé formy testování.

Vývojové, testovací a produkční prostředí jsou na sobě nezávislé, každé má svá data i metadata a často jsou i v rámci infrastruktury společnosti Salesforce provozovány na jiném serveru. Pro nasazení nové funkcionality do provozu je tedy nutné migrovat metadata a data, které vznikly při vývoji, mezi prostředími.

Dle Sommervilla (2013) by mělo být nasazování změn v rámci softwarového inženýrství co nejvíce automatizované. Redukuje se tak riziko chyby ze strany člověka. Dále klade důraz na kvalitní dokumentaci změny pro potřeby operačního týmu, který ji nasazuje, i pro koncové uživatele softwaru. Z dokumentace by mělo být jasné, čeho se změna týká, jaké komponenty jsou součástí a na kdy je či bylo naplánováno změnu

vydat.

Cílem této práce je návrh a implementace systému pro správu a automatizaci vydávání změn v rámci Salesforce CRM. Jedná se o nástroj určený pro administrátory a vývojáře, kteří pracují s platformou Force.com. Snahou je zjednodušit jejich práci, aby při migracích nemuseli kombinovat více softwarových nástrojů a aby ve stejném systému zároveň vznikala dokumentace k vydání změny. Systém je realizován jako webová aplikace, což eliminuje problém s kompatibilitou vůči operačnímu systému a umožňuje efektivní sdílení dat mezi uživateli.

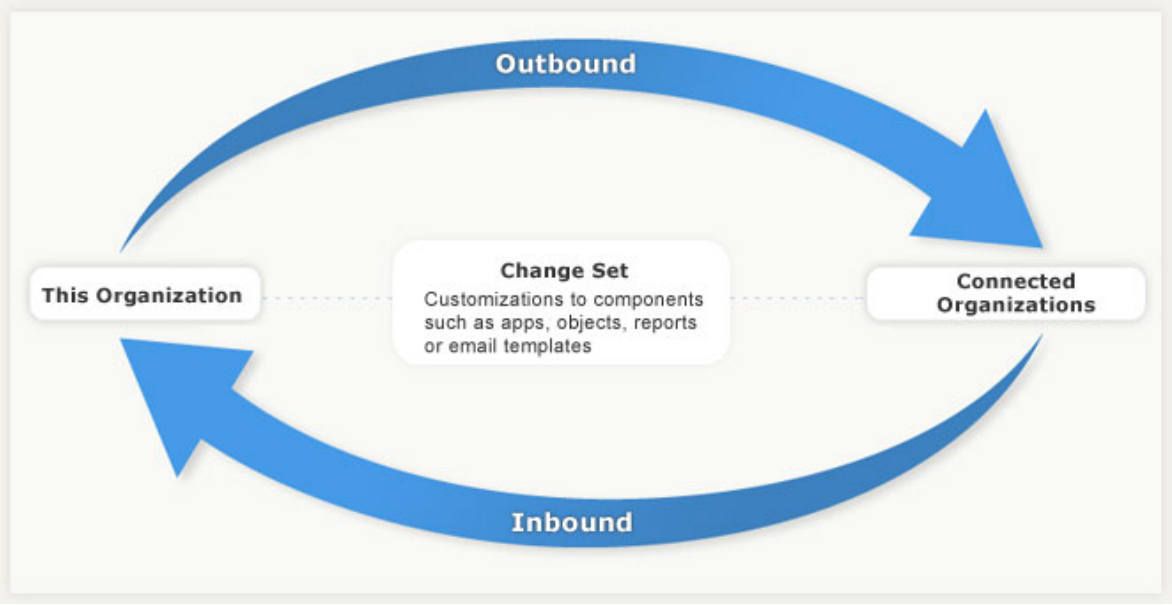
I. TEORETICKÁ ČÁST

1 Analýza současných řešení

Počátky systému Salesforce CRM se datují do roku 1999, což dalo prostor pro vznik hned několika řešením zabývajícím se migrací dat a metadat mezi prostředími. Některé z nich jsou přímo součástí systému nebo alespoň spravovány společností Salesforce. Ostatní jsou produkty jiných společností, kde většinou platí, že pro jejich užití je nutné si zakoupit licenci či produkt předplatit na určitou dobu jako službu. V textu jsou popsány neznámější řešení, jejich výhody a nevýhody a jak ovlivnily vývoj vlastního systému pro vydávání změn.

1.1 Change Sets

Nejdostupnější nástroj pro migraci metadat mezi prostředími se nazývá Change Sets. Je přímo součástí systému Salesforce, bez nutnosti pořizovat dodatečné licence. Před prvním přesunem metadat mezi zvoleným zdrojovým a cílovým prostředím je nejprve nutné povolit migrace mezi těmito prostředími. Pak je možné na zdrojovém prostředí vytvořit tzv. *Outbound change set*, balík komponent, které řeší změnový požadavek. *Outbound change set* se odesílá na cílové prostředí, kde je veden jako *Inbound change set*. Tok dat znázorňuje obrázek (Obr. 1.1) (Development Lifecycle Guide, 2016, s. 3).



Obr. 1.1 Tok dat u nástroje Change Sets

Aby mohla být změna nasazena, je nutné se přihlásit na cílové prostředí a potvrdit změny v *Inbound Change Set*, což může být považováno za zbytečný krok. Velikou nevýhodou je absence možnosti klonovat *Inbound Change Set* na *Outbound Change Set*. Změnový požadavek typicky migruje z vývojového, přes testovací do produkčního prostředí a závisle na procesu může být zapojeno i více typů prostředí. Bez možnosti

klonovat příchozí balík na *Outbound Change Set* je nutné při přesunu na další prostředí znovu vytvářet seznam komponent od začátku. Mimo časovou neefektivitu se zde také otevírá prostor na chyby ze strany člověka.

1.2 Force.com Migration Tool

Další nástroj pro migraci metadat, který poskytuje přímo společnost Salesforce, se jmenuje Force.com Migration Tool. Jedná se o knihovnu implementovanou v jazyce Java, běžící v rámci aplikace Apache Ant (Force.com Migration Tool Guide, 2016, s. 5). Samotný Apache Ant je nástroj pro sestavování aplikací, který je ovládán pomocí textového terminálu či příkazové řádky (The Apache Ant Project, 2018).

Force.com Migration Tool není přímo součástí systému Salesforce, ale lze jej bezplatně stáhnout z webových stránek společnosti. Práce s nástrojem probíhá lokálním počítači a vyžaduje alespoň základní znalosti práce s příkazovou řádkou a konfiguračními soubory, což nemusí vyhovovat některým uživatelům, kteří spoléhají na existenci uživatelského rozhraní.

Nadstavba společnosti Salesforce pro Apache Ant obsahuje funkce pro získání balíku metadat z prostředí systému Salesforce, pro nahrání balíku na určené prostředí, pro spuštění jednotkových testů a další činnosti. Díky tomu, že získaný balík metadat se nachází na lokálním úložišti uživatele, nedochází k problémům s klonováním balíku u migrací přes více prostředí, jako tomu je u nástroje Change Sets.

1.3 Data Import Wizard

Prvním z nástrojů pro migraci dat, který spravuje společnost Salesforce, je Data Import Wizard, který je stejně jako Change Sets součástí CRM systému. Uživatel zde vybírá, do jakého objektu v databázi chce nahrát data, zvolí CSV soubor se záznamy a po namapování sloupců ze souboru na pole objektu se spustí proces nahrávání.

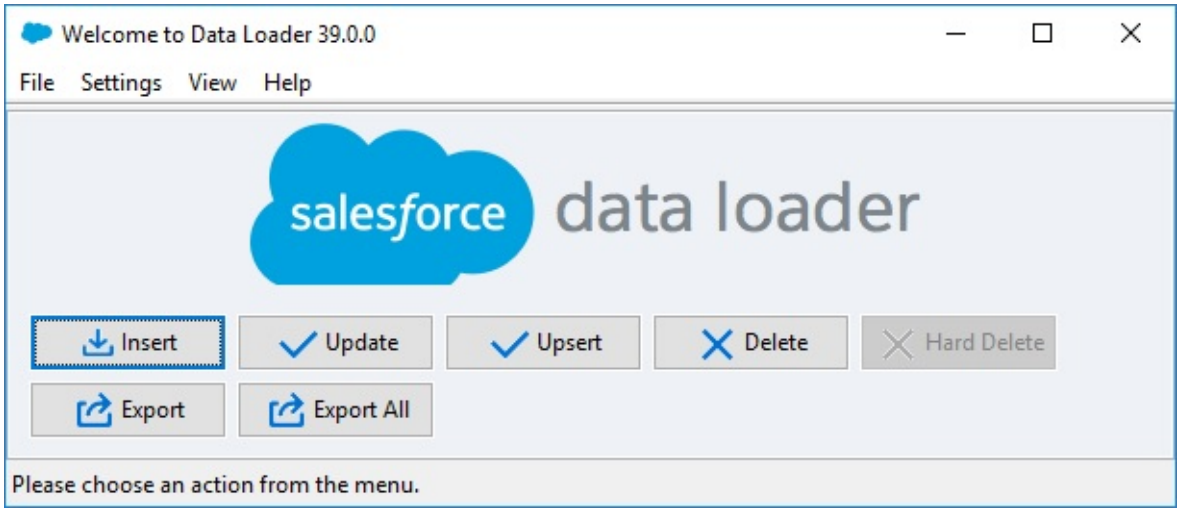
Ovládání nástroje je intuitivní a k používání stačí webový prohlížeč. Aplikace má však své omezení, konkrétně maximum 50 000 záznamů v CSV souboru. Nehodí se tak pro obsáhlé migrace dat. Zároveň postrádá možnost hromadného mazání dat.

1.4 Salesforce Data Loader

Další možnosti, jak migrovat data, je použití aplikace Salesforce Data Loader, která také pochází od společnosti Salesforce. Narozdíl od nástroje Data Import Wizard není součástí samotného systému, ale lze ji dostat zdarma ze stránek společnosti.

Aplikace Salesforce Data Loader také pracuje s CSV soubory, ale již není limitována na počet zpracovaných záznamů. K dispozici jsou všechny typy DML operací - import,

update, upsert i delete (Data Loader Guide, 2016, s. 5). Aplikaci je nutné instalovat a používat na lokálním počítači. Ovládat ji lze přes příkazovou řádku či grafické uživatelské rozhraní, jehož ukázku představuje obrázek (Obr. 1.2).



Obr. 1.2 Uživatelské rozhraní nástroje Salesforce Data Loader

1.5 Gearset

Prvním komerčním nástrojem popisovaným v této práci je aplikace Gearset od stejnojmenné společnosti. Jedná se o webovou aplikaci pro správu migračních plánů a migraci metadat. Kromě manipulace s balíky metadat nabízí porovnání zdrojových kódů mezi prostředími, možnost obnovy staré verze kódu, kompletní historii migrací na spravovaných prostředích a propojení s repozitářem verzovacího systému (Gearset - Features, 2018).

Aplikace je placená na bázi měsíčního předplatného za uživatelskou licenci, což může představovat poměrně vysoký náklad pro vícečlenné vývojové týmy. Navíc se řešení zabývá pouze migrací metadat, chybí možnost manipulace s daty.

1.6 Copado

Dalším komerčním řešením pro vydávání změn do systému Salesforce je aplikace Copado od společnosti Copado Solutions. Taktéž se jedná o webovou aplikaci, ale narozdíl od Gearset se distribuuje jako rozšíření pro Salesforce CRM.

Stejně jako Gearset umí Copado manipulovat pouze s metadaty. Aplikace vyžaduje propojení prostředí systému Salesforce s větvemi kódu v rámci repozitáře verzovacího systému, což vede ke kvalitní správě verzí na platformě Force.com. Navíc je zde možnost spravovat změnové požadavky přímo v aplikaci a do procesu vydávání změn lze také přidat automatické spouštění testovacích procedur (Copado - Services, 2018). Nevýho-

dou je opět dodatečný náklad v podobě měsíčního předplatného za licenci pro jednoho uživatele.

1.7 Shrnutí

Každé řešení má své výhody a nevýhody a hodí se pro specifické případy užití. Cílem výsledného systému této práce je poučit se ze spatřených nedostatků a inspirovat se pozitivními vlastnostmi zkoumaných nástrojů.

Výsledný systém pro vydávání změn je realizován jako webová aplikace, která podporuje manipulace s daty a metadaty. Podporuje i další opomíjené migrační aktivity - spouštění Apex skriptů na cílovém prostředí a manuální kroky. Na datové migrace nejsou aplikovány omezení jak u nástroje Data Import Wizard. Z nástroje Change Sets si bere inspiraci v uživatelském prostředí, z Force.com Migration Tool pak způsob manipulace s balíky metadat. Systém eviduje migrační plány a poskytuje historii migrací.

2 Zvolené technologie

Jelikož je výsledný systém realizován jako webová aplikace, bylo před započetím implementace nutné stanovit, jaké se použijí technologie na straně serveru a na straně klienta. Při výběru rozhodovalo několik kritérií, přičemž nejdůležitější je způsobilost technologie pro implementaci systému daných parametrů. Dále se hledělo na cenu licence, kompatibilitu s různými operačními systémy, kvalitu dokumentace a také osobní preference a zkušenosti autora práce.

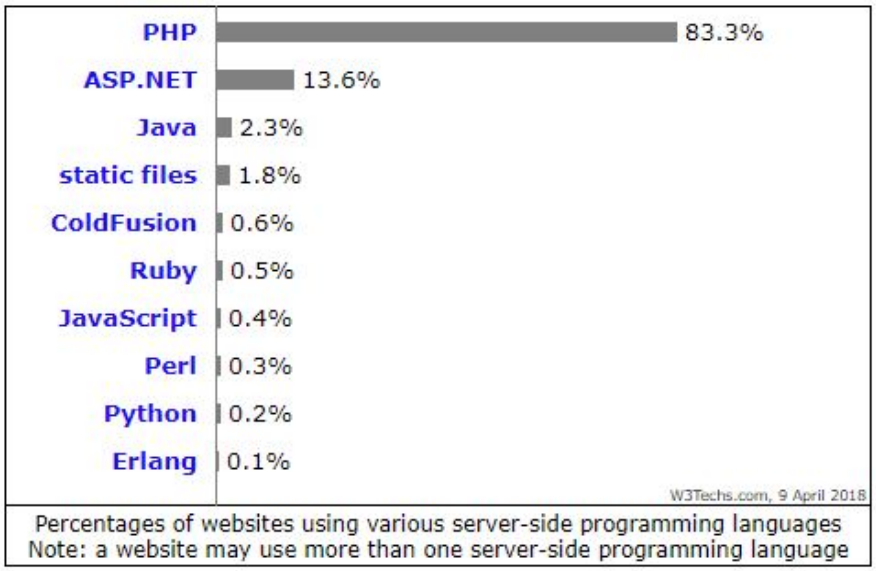
2.1 Technologie na straně serveru

Serverová strana systému odpovídá na požadavky klienta, který je v případě webové aplikace internetový prohlížeč uživatele, a uchovává data uživatelů. Již před počátkem implementace bylo jasné, že technologie na straně serveru musí také zvládat komunikaci se vzdáleným systémem (Salesforce) a poskytovat efektivní cestu k autentizaci a autorizaci uživatelů, aby zpřístupnila data jen oprávněným subjektům.

První zvažovanou možností bylo postavení aplikace na platformě Force.com, tedy přímo v rámci systému Salesforce. Výhodou by byla možnost využití již existujícího řešení pro autentizaci uživatelů. Platforma Force.com však byla zamítnuta pro své omezení v přístupu do souborového systému na serveru a další limitace. Možnosti programovacího jazyku Apex jsou více vhodné pro rozšiřování standardních funkcionalit systému Salesforce.

Při dalším hledání byly vzaty v úvahu výzkumy používanosti a oblíbenosti serverových technologií. Graf v obrázku (Obr. 2.1), který pochází od společnosti W3Techs, znázorňuje, že nejvíce zastoupený je jazyk PHP, následovaný technologií ASP.NET a Javou (Usage of server-side programming languages for websites, 2018). Společnost TIOBE dělá pravidelné výzkumy oblíbenosti programovacích jazyků. Obrázek (Obr. 2.2) uvádí umístění z ledna 2018 (TIOBE Index, 2018).

Pro implementaci výsledného systému by šlo využít jakéhokoliv z vysoce umístěných jazyků a technologií. Rozhodnuto bylo pro technologii ASP.NET s využitím programovacího jazyku C#. Hlavní roli hrála osobní preference a zkušenost autora práce. Aspekty podporující tuto volbu jsou obsáhlá dokumentace, početná komunita pro diskuzi nad problémy, rozšířitelnost standardních komponent a funckionalit, dokonalý systémem routování, moderní aplikační rozhraní a další důvody, které uvádí Freeman (2017, s.6–9). Konkrétně se ve výsledném systému používá verze ASP.NET Core 2.



Obr. 2.1 Výzkum zastoupenosti programovacích jazyků na straně serveru

Jan 2018	Jan 2017	Change	Programming Language
1	1		Java
2	2		C
3	3		C++
4	5	^	Python
5	4	v	C#

Obr. 2.2 TIOBE Index z ledna roku 2018

2.1.1 ASP.NET Core 2

Technologie ASP.NET pochází od společnosti Microsoft a její počátky se datují do roku 2002. Jedná se o rozšíření předchozí skriptovací platformy ASP na komplexnější technologii určenou pro dynamické zpracovávání webových stránek na straně serveru. Technologie od té doby prošla mnoha obměnami, kde poslední výraznou změnou je uvedení ASP.NET Core v roce 2015 (Freeman, 2017, s. 3).

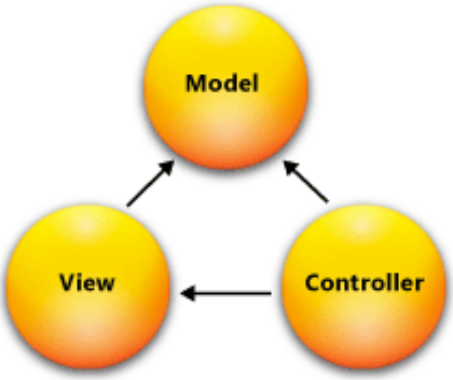
Framework ASP.NET Core je narozdíl od svých předchůdců distribuován jako *open-source*, tedy nabízí možnost nahlédnout a podílet se na zdrojovém kódu technologie, a také již nevyžaduje provoz na operačním systému Microsoft Windows. Vývoj a používání výsledných aplikací je možný i na operačním systému macOS od společnosti Apple a na různých distribucích systému Linux (Freeman, 2017, s. 6). Dále .NET Core přináší vlastní nástroj .NET Core CLI, ovládaný přes příkazovou řádku, který usnadňuje práci s vývojem a sestavováním aplikací.

Nejnovější verze ASP.NET Core 2 obohacuje první verzi o některé chybějící aplikační rozhraní z původních verzí ASP.NET. Rozšiřuje také podporu více distribucí operačního systému Linux. Největším přínosem této verze je možnost využití Razor Pages. Jedná se o nový, modernější způsob rozmístění logiky kódu (Freeman, 2017, s. 6).

Architekturu webové aplikace implementovanou s pomocí frameworku ASP.NET Core 2 lze postavit pojmout dvěma způsoby. První možností je MVC architektura, kde framework zasahuje i do vizuální části aplikace. Druhý způsob se nazývá Web API a uvažuje pouze implementaci webových služeb na straně serveru, které jsou volány klientem za použití jakékoliv jiné vhodné technologie. Pro výsledný systém této práce byla zvolena MVC architektura.

2.1.2 Architektura MVC

Jak již bylo avizováno, architektura MVC je jedním z přístupů, jak postavit aplikaci za použití ASP.NET Core a nejen této technologie. MVC je zkratkou pro Model-View-Controller, kde jednotlivé části složeniny představují vrstvu aplikace. Obecně architektura odděluje prezentaci a interakci od systémových dat (Sommerville, 2013, s. 155). Dle Freemana (2017, s. 6) implementace MVC v ASP.NET vysoce zlepšila separaci logiky v rámci kódu oproti předchozím architekturám. Obrázek (Obr. 2.3) znázorňuje, jak spolu jednotlivé vrstvy interagují.



Obr. 2.3 Interakce mezi vrstvami v architektuře MVC

Vrstva Model Modely jsou objekty, které implementují logiku pro práci s daty. Vrstva se často používá k reflektování stavu databáze, kde třídy modelu odpovídají tabulkám schématu databáze (ASP.NET MVC Overview, 2012).

Vrstva View Vrstva View obsahuje komponenty, které se starají o zobrazení uživatelského prostředí. Využívá technologií typických pro stranu klienta - značkovací jazyk

HTML s podporou kaskádových stylů CSS, skriptovací jazyk JavaScript a v případě ASP.NET i pre-procesor Razor.

Razor je technologie na straně serveru, která slouží jako nástroj pro efektivní generování HTML kódu. Nabízí možnost dynamicky generovat obsah z dat na serveru. Jedná se výchozí view engine pro aplikace psané v ASP.NET. Pro zápis kódu lze použít syntax vycházející z jazyku C# či Visual Basic (Freeman, 2017, s. 105). Ukázku technologie Razor, kde je na základě dat ze serveru generován seznam produktů, znázorňuje obrázek (Obr. 2.4).

```
<ul>
    @foreach(var product in Model.Products)
    {
        <li>@product.Name</li>
    }
</ul>
```

Obr. 2.4 Ukázka zdrojového kódu technologie Razor

Vrstva Controller Prostředníkem mezi vrstvou View a Model je vrstva Controller. Komponenty v ní zpracovávají vstup uživatele, pracují s modely a připravují výstup. U webových aplikací představují vstupy uživatele HTTP požadavky. Technnologie ASP.NET na základě cílové URL adresy a metody z požadavku rozezná, která třída a akce z vrstvy Controller se má postarat o odpověď, případně vrací chybový stav (Add a controller to an ASP.NET Core MVC app, 2017).

2.1.3 Databáze

Kromě manipulací s daty a metadaty z systému Salesforce pracuje výsledný systém s vlastními daty. Nejvhodnějším způsobem, jak uchovávat data na serveru, je v databázovém systému, který poskytuje možnosti zabezpečení a řeší problémy s integritou, redundancí a nekonzistencí dat (Pokorný a Valenta, 2013).

Pro výsledný systém byl vybrán relační databázový systém PostgreSQL ve verzi 10. PostgreSQL je distribuován jako open-source, nicméně zvládá funkce jiných komerčních řešení. Nabízí možnost vytvoření procedur a agregačních funkcí přímo v rámci databázového systému a to ve škále různých programovacích jazyků. Poskytuje také rozhraní pro vytvoření vlastních datových typů včetně operací, které nad nimi lze provádět. Výhodou je také možnost provozovat databázový server na všech známých operačních systémech - Unix, Linux, macOS i Microsoft Windows (Obe a Hsu, 2015).

2.1.4 Entity Framework

Pro ulehčení komunikace s databází se ve výsledné aplikaci využívá technologie Entity Framework. Jedná se o nástroj, který je schopný propojit tabulky v databázi s doménovými třídami systémy, v tomto případě z vrstvy Model. Šetří tak čas a řádky zdrojového kódu, které by se musely investovat do konstrukce SQL dotazů, zpracovávání výsledků dotazů a převádění získaných struktur do vlastních objektů (Introduction to Entity Framework, 2016).

Práci s Entity Framework lze pojmout dvěma směry. Záleží, jestli k vyvíjené aplikaci již existuje databáze či nikoliv. Pokud tvorba a plnění databáze předcházely vývoji aplikace, používá se nástroj Entity Framework způsobem *database first*. Tabulky v databázi se zde podrobují reverznímu inženýrství, kde jsou zkoumány atributy, omezení, vztahy a další vlastnosti schématu. Využít lze nástrojů ve vývojovém prostředí Visual Studio, které se připojí k databázi a ze schématu vytvoří třídu, jež je potomkem třídy DbContext a představuje rozhraní pro další komunikaci s databází, a třídy odpovídající tabulkám schématu (Entity Framework Database First, 2016).

Jednodušší situace nastává, pokud databázové schéma vzniká zároveň s aplikací. V takovém případě lze využít postupu *code first*, kde se nejprve implementují doménové třídy v rámci kódu, které se pak pomocí nástrojů technologie Entity Framework reflektují do databázového systému jako tabulky schématu. Pomocí speciálních anotací na atributech tříd lze nastavit, zda je atribut v rámci objektu/tabulky unikátní, jakou má mít maximální délku, zda má mít v tabulce jiný název, že obsah povoluje jen určité znaky a další vlastnosti. Anotace jsou brány v úvahu při konverzi objektů na tabulky stejně tak, jako vztahy mezi objekty (Entity Framework Code First to a New Database, 2016).

Ve výsledném systému této práce se využívá technologie Entity Framework ve verzi 2.0. Jelikož návrh a implementace doménového modelu předcházely práci na databázovém systému, bylo postupováno způsobem *code first*.

2.2 Technologie na straně klienta

Ve výsledném systému se využívá značkovacího jazyka HTML pro strukturu prezentovaného dokumentu, kaskádových stylů CSS pro úpravu vzhledu stránky a skriptovacího jazyka JavaScript pro dodatečné interakce na podněty uživatele v rámci uživatelského prostředí. Pro ulehčení práce bylo využito dalších dvou knihoven - Boostrap a jQuery.

2.2.1 Bootstrap

Bootstrap je sada nástrojů usnadňující vývoj webových aplikací. Její počátky se datují do roku 2011, kdy vznikla ve společnosti Twitter jako reakce na potřebu standardizovat vývoj vzhledu aplikace (Spurlock, 2013).

Boostrap nabízí již implementované styly, které lze aplikovat na HTML elementy. Jedná se o různé vzhledy tlačítek, panelů, oken, menu, tabulek, vstupních polí, ikon a dalších vizuálních prvků. Lze tak snadno vytvořit moderně vypadající web s minimálním úsilím (Spurlock, 2013).

Framework je distribuován jako *open-source* a existuje již ve více verzích. V době psaní této práce byla nejaktuálnější verze 4.1. Pro kompatibilitu s většinou šablon webových stránek je ve výsledném systému zakomponována verze 3.

2.2.2 jQuery

Knihovna jQuery taktéž nabízí prostředky pro urychlení vývoje aplikací na straně klienta. Je postavena na jazyku JavaScript. Přináší do skriptování na straně klienta syntax čerpající z výhod selektorů kaskádových stylů CSS a jednodušší zápis některých funkcí (Chaffer, 2013). Mezi hlavní případy užití knihovny jQuery patří přístup k elementům v HTML dokumentu, dynamická úprava vzhledu stránky, dynamické změny obsahu elementů, různé animace, asynchronní volání webových služeb a další akce, které by při použití klasického JavaScriptu zabraly více řádků zdrojového kódu (Chaffer, 2013). Z výše zmíněných důvodů je knihovna zakomponována i do výsledného systému této práce, konkrétně ve verzi 2.2.

Knihovna jQuery u některých typů elementů doplňuje funkcionality knihovny Boostrap. K vzhledu prvků přidává i reakce na podněty uživatele. Příkladem v této práci jsou například modální okna, které z knihovny Boostrap přebírají kaskádové styly a z knihovny jQuery možnost se dynamicky ukázat či skrýt v rámci uživatelského prostředí.

2.3 Technologie pro migraci dat a metadat

Následující text popisuje prostředky použité pro migraci dat a metadat mezi prostředími systému Salesforce. Jedná se o aplikace či aplikační rozhraní, které poskytuje samotná společnost Salesforce. Bylo posuzováno, zda svými možnostmi mohly přispět k vývoji výsledného systému.

2.3.1 Salesforce SOAP API

Salesforce SOAP API je název pro webové aplikační rozhraní, které využívá protokolu SOAP. Klient a server spolu zde komunikují pomocí XML dokumentů s jasně definovanou strukturou. Aplikační rozhraní nabízí funkce pro vytvoření, upravení či získání záznamů v databázi, funkce pro získání vlastností z různých částí systému Salesforce a mimo jiné i možnost autentizovat uživatele (SOAP API Developer Guide, 2016).

Pro ověření identity uživatele se používá akce *login()*. Klient v požadavku posílá uživatelské jméno, heslo a typ prostředí systému Salesforce. Server při úspěšné validaci vstupních dat vrací v odpovědi přístupový token s časem jeho expirace (SOAP API Developer Guide, 2016). Pro jednoduchost uvedeného způsobu autentizace je volání aplikačního rozhraní Salesforce SOAP API použito ve výsledném systému práce.

Aplikační rozhraní by teoreticky mohlo sloužit i k manipulaci s daty. Jsou k tomu určené akce *create()*, *update()*, *upsert()* a *delete()* (SOAP API Developer Guide, 2016). Tato možnost však byla zamítnuta kvůli limitacím na objemu zpracovaných záznamů v jednom HTTP požadaku. V jedné transakci lze totiž vytvořit či upravit pouze 200 záznamů (SOAP API Call Limits, 2018).

2.3.2 Salesforce Metadata API

Dalším zkoumaným aplikačním rozhraním bylo Salesforce Metadata API. Jak už název napovídá, rozhraní slouží k manipulaci s metadaty na prostředí systému Salesforce. Nabízí funkce pro vytvoření, upravení, smazání, stáhnutí či nahrání metadat. Pomocí Salesfoce Metadata API není možné vytvořit záznamy do objektu v databázi, k čemuž slouží jiné aplikační rozhraní, lze však např. vytvořit samotný objekt (Metadata API Developer Guide, 2016).

Výsledný systém využívá většinu výše zmíněných akcí aplikačního rozhraní. Volá akci *retrieve()* pro spuštění procesu získávání metadat a akci *deploy()* pro spuštění nahrávání balíku metadat. Akce *checkRetrieveStatus()* a *checkDeployStatus()* pak slouží ke kontrole stavu spuštěných manipulací.

2.3.3 Force.com Migration Tool

K manipulaci s metadaty připadá v úvahu i již avizovaný nástroj Force.com Migration Tool. Výhodou by byla snazší integrace do systému. Jelikož je nástroj sestrojen jako konzolová aplikace, bylo by dostatečné ji v systému spustit jako podproces a konzumovat výsledky z výstupu. Tento návrh implementace byl však zamítnut, jelikož by v porovnání s voláním Salesforce Metadata API dosahoval horších výsledků z hlediska výpočetního výkonu a konzumace paměti serveru. Tvrzení potvrzuje fakt, že nástroj

Apache Ant je napsán v jazyce Java, který pro své aplikace vyžaduje mít spuštěné své běhové prostředí (JRE).

2.3.4 Salesforce REST API

Dále bylo zkoumámo aplikační rozhraní Salesforce REST API. Jak už název napovídá, rozhraní využivá REST architektury, která je postavena na zdrojích (např. objekt *Product* v rámci databáze) s přidělenou adresou URL. Na tuto adresu se pak typicky posílají požadavky metodami GET, POST, PATCH a DELETE pro získání, vytvoření, úpravu a smazání zdroje (Mumbaikar a Padiya, 2013).

Salesforce REST API nabízí akce pro manipulaci s daty, stejně jak Salesforce SOAP API. Obě rozhraní sdílí také stejný limit na počet zpracovaných souborů v rámci jednoho HTTP požadavku. Z tohoto důvodu bylo použití rozhraní ve výsledné práce zamítnuto, i přes výhody plynoucí z REST architektury. Rozhodnutí podpořil fakt, že obsah těl požadavků nemůže být strukturován jako CSV soubor. Rozhraní podporuje pouze formát XML či JSON (Force.com REST API Developer Guide, 2016).

2.3.5 Salesforce Bulk API

Další zkoumané aplikační rozhraní pro manipulaci s daty se nazývá Salesforce Bulk API. Slovo „bulk“ v angličtině znamená „hromadný“ či „masa“ a rozhraní opravdu slouží ke zpracovávání velkého objemu dat. Rozhraní funguje na principech REST architektury (Bulk API Developer Guide, 2016).

Princip manipulace s daty pomocí Bulk API rozhraní se liší od výše popsaného REST API. Spočívá ve vytvoření a odeslání HTTP požadavků pro každou z následujících akcí:

1. vytvoření úkolu pro hromadné zpracování dat,
2. vytvoření dávek s daty,
3. spuštění úkolu pro hromadné zpracování dat,
4. a získání výsledku po dokončení úkolu (Bulk API Developer Guide, 2016).

V původní verzi Salesforce Bulk API bylo pro vykonání druhého kroku nutné rozdělit zdrojový soubor na více dávek pomocí vlastních prostředků a každou dávku odeslat v samostatném HTTP požadavku. Od října roku 2017 tohle úskalí řeší nová verze rozhraní, nazývána Bulk API 2.0 (Salesforce.com Winter ’18 Release Notes, 2017).

Nová verze aplikačního rozhraní nepožaduje samostatný požadavek pro každou dávku dat. Po vytvoření úkolu pro hromadné zpracování je možné poskytnout data v samostatném požadavku. Limit na velikost požadavku činí 20 000 záznamů, jsou-li data

odesílány jako soubor, a 150 MB, když jsou data reprezentovány přímo v textové podobě (Bulk API 2.0, 2017).

Pro způsob manipulace s velkými soubory bylo aplikační rozhraní Bulk API 2.0 zvoleno jako vhodná technologie pro migraci dat mezi prostředími systému Salesforce. Dalším důvodem je podpora formátu CSV.

2.3.6 Tooling API

Poslední popisované aplikační rozhraní v této práci nese název Tooling API, které taktéž pochází od společnosti Salesforce. Doplňuje některé z funkcí Salesforce Metadata API a používá se pro práci s detaily již vytvořených metadat. Požadavky lze posílat způsobem architektury REST i SOAP. Rozhraní nabízí akce pro získání informace o objektu včetně jeho polí, získání záznamů z vývojářského logu, spouštění testů, výpočet pokrytí kódu testy a další činnosti (Tooling API, 2016). Pro potřeby této práce bylo Tooling API začleněno pro možnost spouštění Apex skriptů v systému.

3 Bezpečnost webové aplikace

Podle výzkumu provedeným společností Cybersecurity Ventures byly škody způsobené kybernetickými útoky v roce 2015 vyčísleny na 400 až 500 biliónů amerických dolarů. Každým rokem se zvyšuje i celková částka investovaná do bezpečnosti IT a odhaduje se, že v součtu za roky 2017 až 2021 přesáhne hodnotu jednoho trilionu amerických dolarů (Cybersecurity Market Report, 2017). Dle LeBlanca a Messerschmidta (2016) jsou však navzdory investicím téměř každý týden hlášeny úniky citlivých informací o uživatelích, jako např. přístupová hesla, čísla kreditních karet či zdravotní záznamy.

Problematika bezpečnosti softwarových produktů byla brána v úvahu i během analýzy a vývoje výsledného systému této práce. Následující text popisuje známé hrozby, které mohou narušit integritu webových aplikací včetně možností, jak jim předejít a ochránit tak výsledný produkt.

3.1 Autentizace uživatelů

Autentizace uživatelů je důležitou součástí webových aplikací, kde má uživatel možnost se přihlásit a pracovat se svými daty. Autentizací se rozumí proces ověřování identity uživatele (LeBlanc a Messerschmidt, 2016). Je nutné rozlišovat autentizaci od autorizace.

Ověření totožnosti uživatele může proběhnout více způsoby. Záleží na možnostech použité technologie. U webových aplikací se uživatel nejčastěji prokazuje něčím, co zná, zpravidla uživatelským jménem a heslem. Dále lze totožnost ověřit pomocí něčeho, co uživatel vlastní, například čipovou kartou či certifikátem, nebo pomocí biometrických údajů (otisk prstu, snímek oční duhovky či sítnice) (Přibyl, 2009).

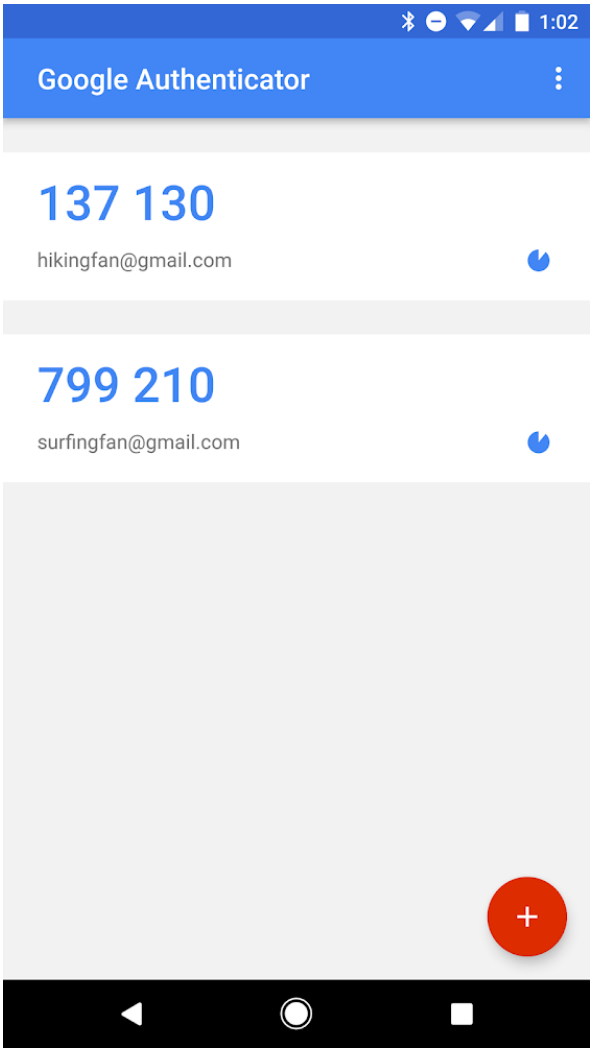
3.1.1 Vícefaktorová autentizace

Z důvodu zvýšení bezpečnosti lze výše zmíněné metody autentizace kombinovat. Zavádí se tak do systému tzv. vícefaktorová autentizace. Ověřování více faktorů zajišťuje dodatečnou ochranu v případě, je-li jedna z ochran totožnosti prolomena (Mueller, 2016).

Nejčastější kombinací je přihlašování pomocí uživatelského jména a hesla společně s něčím, co uživatel má. Oblibě se těší druhotné ověřování pomocí dočasných, jednorázových hesel generovaných přímo pro uživatele a službu, vůči které se autentizuje (LeBlanc a Messerschmidt, 2016). V takovém případě se jedná o dvoufaktorovou autentizaci, zkráceně 2FA.

Existuje více způsobů generování jednorázových, dočasných hesel. Nejoblíbenější z nich je založen na časové synchronizaci. Spočívá v generování kódů s krátkou do-

bou platnosti, po kterou ho lze použít. Tajemství může být uživateli doručeno např. pomocí SMS či emailové zprávy. Závisle na možnostech používané služby lze dočasný, jednorázový kód získat také pomocí mobilních aplikací, určených pro dvoufaktorovu autentizaci. Jednou z nich je např. Google Authenticator (LeBlanc a Messerschmidt, 2016). Ukázku aplikace znázorňuje obrázek (Obr. 3.1) (Google Authenticator, 2018).



Obr. 3.1 Ukázka aplikace Google Authenticator

3.1.2 ASP.NET Core Identity

Ve výsledném systému této práce se totožnost uživatele ověřuje na základě poskytnutého uživatelského jména a hesla. Dle Přibyla (2009) je tato metoda i přes svoji jednoduchost poměrně bezpečná. Systém využívá funkce technologie ASP.NET Core Identity.

Jedná se o aplikační rozhraní od společnosti Microsoft, které slouží ke správě uživatelů v aplikacích postavených na ASP.NET. Umožňuje vytvoření, úpravu a smazání

uživatele za pomocí již popsané technologie Entity Framework. Nedílnou součástí rozhraní je možnost autentizace uživatele (Freeman, 2017). Pro proces ověření totožnosti lze nastavit různé parametry, např. požadavky na složitost hesla. Další možností je autentizace uživatele pomocí alternativních poskytovatelů identity, mezi které patří Facebook, Google, Microsoft Account, Twitter a další autority (Introduction to Identity on ASP.NET Core, 2018).

3.1.3 Autentizace vůči systému Salesforce

Jelikož výsledný systém komunikuje s prostředími systému Salesforce, bylo nutné analyzovat způsoby autentizace vůči této entitě. Přihlašení do systému Salesforce podmiňuje poskytnutí uživatelského jména a hesla. V nastavení profilu uživatele lze omezit přihlášení jen na určité IP adresy a pracovní dobu. Salesforce taktéž nabízí škálu možností pro zavedení dvoufaktorové autentizace (Salesforce Security Guide, 2016).

Ve výchozím nastavení je dvoufaktorového ověření vyžadováno, přihlašuje-li se uživatel z IP adresy, která se nachází mimo rozsah povolených IP adres. V takové situaci musí uživatel mimo uživatelského jména a hesla poskytnout další ověřovací kód. Přihlašuje-li se uživatel do systému klasicky pomocí webového uživatelského rozhraní, vyžaduje systém poskytnutí dočasného kódu, který byl odeslán v emailové zprávě či pomocí SMS. Přistupuje-li uživatel k funkcím systému Salesforce z jiné aplikace pomocí webového aplikačního rozhraní, je nutné poskytnout speciální bezpečnostní token. Jedná se o klíč, který je generován systémem Salesforce a přidělen uživateli při nastavení či každé změně hesla. Uživatel poskytuje bezpečnostní token při autentizaci vůči prostředí systému Salesforce jako součást svého hesla (Salesforce Security Guide, 2016).

Pro přístup k funkcím systému již není potřeba bezpečnostního tokenu, nýbrž přístupového tokenu, který uživatel obdrží při úspěšné autentizaci. Tento klíč představuje identitu uživatele a je platný pouze po omezenou dobu, kterou lze v systému nastavit. Po skončení platnosti systém vynucuje dodatečné ověření totožnosti. Znalost popsaného mechanismu byla klíčová pro implementaci výsledného systému, jelikož využívá funkce aplikačních rozhraní Salesforce SOAP API, Salesforce Metadata API a dalších již popsaných služeb.

3.2 Autorizace uživatelů

Zatím co se autentizace soustředí na ověřování identity uživatele, autorizace řeší problematiku přístupu k chráněným zdrojům. Ve většině případů autorizace navazuje na autentizaci. Její podstatou je ověření, zda má uživatel oprávnění provést určitou akci, např. vytvořit záznam ve zvolené tabulce. Autorizace se často řeší vytvořením rolí či profilů, které se přiřadí uživatelům. Vznikají tak skupiny uživatelů s různými oprávn-

něními, např. administrátoři, správci určité oblasti, bežní uživatelé apod. (LeBlanc a Messerschmidt, 2016).

3.2.1 Autorizace v ASP.NET Core

Stejně jak v případě autentizace, i pro autorizaci nabízí technologie ASP.NET Core již hotové rozhraní. Práce vývojáře pak spočívá ve výběru a implementaci vhodného typu autorizace. Ověření přístupu může probíhat na základě role či oprávnění uživatele (Introduction to authorization in ASP.NET Core, 2016).

Autorizace na základě role uživatele V systému lze vytvořit jakoukoliv uživatelskou roli - *Administrator*, *PowerUser*, *HRManager* apod. Název a rozsah její možností stanovuje sám vývojář. Taktéž není vyloučeno, že uživatel může disponovat více rolemi (Role-based authorization in ASP.NET Core, 2016).

Autorizace na základě rolí funguje na jednoduchém principu. Vývojář pomocí speciální anotace označí, které role smí používat určenou třídu či konkrétní akce ve třídě. Obrázek (Obr. 3.2) (Role-based authorization in ASP.NET Core, 2016) ilustruje příklad, kde role *Administrator* a *PowerUser* mají přístup ke všem akcím třídy *ControlPanelController* kromě akce *ShutDown()*, kterou smí spouštět pouze uživatel s rolí *Administator*.

```
[Authorize(Roles = "Administrator, PowerUser")]
public class ControlPanelController : Controller
{
    public ActionResult SetTime()
    {
    }

    [Authorize(Roles = "Administrator")]
    public ActionResult ShutDown()
    {
    }
}
```

Obr. 3.2 Ukázka autorizace na základě rolí

Autorizace na základě oprávnění Anglické spojení *policy-based authorization* bylo pro účel této práce přeloženo jako autorizace na základě oprávnění. Ověření zde funguje na principu předem definovaných oprávnění, jejichž parametry musí uživatel splnit, chce-li přistoupit ke zdroji (Policy-based authorization in ASP.NET Core, 2017).

Nejjednodušší implementace autorizace s využitím oprávnění funguje na základě ověření tvrzení. Tvrzení představuje dvojice jméno a hodnota, např. řetězec *Employe-*

eNumber (identifikátor zaměstnance) a pořadové číslo zaměstnance. Tvrzení lze pak využít pro vytvoření oprávnění, které dává přístup pouze pro zaměstnance. Obrázek (Obr. 3.3) znázorňuje vytvoření oprávnění uvedeného příkladu (Claims-based authorization in ASP.NET Core, 2016).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("EmployeeOnly", policy => policy.RequireClaim("EmployeeNumber"));
    });
}
```

Obr. 3.3 Ukázka autorizace na základě tvrzení bez hodnoty

V zdrojovém kódu lze vidět deklaraci oprávnění *EmployeeOnly*, které vyžaduje, aby bylo splněno tvrzení *EmployeeNumber*. Pro přístup ke zdroji zde stačí, že má uživatel přiřazeno jakékoliv zaměstnanecké číslo, nezkoumá se hodnota tvrzení. Obrázek (Obr. 3.4) pak ilustruje vytvoření přístupu jen pro určité zaměstnance, konkrétně pro zakladatele společnosti. Tvrzení nese název *Founders* (zakladatelé). Zde se už při vyhodnocení udělení přístupu zkoumá hodnota zaměstnaneckého čísla (Claims-based authorization in ASP.NET Core, 2016).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddAuthorization(options =>
    {
        options.AddPolicy("Founders", policy =>
                          policy.RequireClaim("EmployeeNumber", "1", "2", "3", "4", "5"));
    });
}
```

Obr. 3.4 Ukázka autorizace na základě tvrzení s hodnotami

Autorizace na základě oprávnění může využívat i více komplexní logiku, která však vyžaduje více kódu. Je nutné vytvořit třídu, která nese parametry zkoumaného oprávnění, např. *MinimumAgeRequirement* pro reprezentaci požadavku na minimální věk, a třídu zpracovávající oprávnění, např. *MinimumAgeHandler* pro vyhodnocení dostatečnosti věku u určeného uživatele. Výhodou zmíněného přístupu je zvýšená znovupoužitelnost kódu a lepší možnosti pro testování (Policy-based authorization in ASP.NET Core, 2017).

3.3 Cross-site Request Forgery

Cross-site Request Forgery, zkráceně CSRF, je typ útoku na webovou aplikaci, kde útočník spouští akce na serveru pod identitou a právy jiného uživatele. Útok zpravidla není využíván k získání dat, jelikož útočník nevidí výsledek útoku a dokáže pouze spustit nechráněnou akci. Je však žádoucí se proti CSRF bránit (Mueller, 2016).

Provedení útoku vyžaduje znalost cílové webové aplikace, konkrétně URL adresu a podobu požadavku, přes který lze spustit zvolenou akci. Poté stačí uživateli předat odkaz na vlastní webovou stránku, která akci spustí pod jeho totožností. Často je využíváno technik sociálního inženýrství, např. předání odkazu pomocí falešné emailové zprávy. Úspěšnému útoku musí předcházet zdárná autentizace a autorizace oběti (Cross-Site Request Forgery, 2018).

Proti CSRF se lze chránit více způsoby. Pokud webová aplikace umožňuje nastavení restrikce na původ HTTP požadavku, implementuje se na straně serveru kontrola hlavičky *Referer*. Hodnota hlavičky obsahuje URL adresu, ze které je požadavek odeslán. Server pak má možnost vyhodnotit, zda se hodnota shoduje s adresou URL, na které je provozována webová aplikace. Bylo však dokázáno, že hodnotu hlavičky *Referer* lze podvrhhnout, tudíž je nutné tento způsob ignorovat nebo kombinovat s dalšími bezpečnostními praktikami (Cross-Site Request Forgery, 2018).

Častou ochranou je zavedení synchronizačních tokenů (CSRF tokenů). Token se generuje na straně serveru jako unikátní klíč pro relaci uživatele s dostatečnou mírou náhodnosti. Uživateli je zpravidla předán jako hodnota skrytého pole ve formuláři. Při spouštění akce musí být token součástí HTTP požadavku, aby jej server mohl přečíst, vyhodnotit, zda je validní, a bránit se tak proti CSRF (Cross-Site Request Forgery, 2018).

3.3.1 Ochrana proti CSRF v ASP.NET Core

Technologie ASP.NET Core nabízí možnost ochrany proti CSRF pomocí již popsaných CSRF tokenů. Akce, které chce vývojář zabezpečit, stačí obohatit o anotaci *ValidateAntiForgeryToken*. Příklad zdrojového kódu znázorňuje obrázek (Obr. 3.5). Při spouštění akce server automaticky zkoumá přítomnost a hodnotu tokenu v HTTP požadavku (Prevent Cross-Site Request Forgery attacks in ASP.NET Core, 2018).

Na straně klienta je CSRF token generován automaticky jako skryté pole v každém formuláři. Není-li HTTP požadavek odeslán pomocí formuláře, ale např. za použití technologie AJAX, lze token získat pomocí speciálního makra *@Html.AntiForgeryToken()* (Prevent Cross-Site Request Forgery attacks in ASP.NET Core, 2018).

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Logout()
{
    await _signInManager.SignOutAsync();
    _logger.LogInformation("User logged out.");
    return RedirectToAction(nameof(HomeController.Index), "Home");
}
```

Obr. 3.5 Ochrana ASP.NET Core proti CSRF na straně serveru

3.4 Cross-site Scripting

Další analyzovaná hrozba webové aplikace se nazývá Cross-site Scriptiing, často vedená pod zkratkou XSS. Princip útoku spočívá v zneužití nedostatečného ošetření vstupů do aplikace, přes které útočník vkládá vlastní zdrojový kód způsobující nechtěné chování aplikace. Vkládaný kód je zapsán v jazyku, který interpretuje prohlížeč na straně klienta, nejčastěji v jazyku JavaScript (Cross-site Scripting, 2018). Následky spuštění škodlivého kódu mohou být různé. Může dojít např. k přesměrování uživatele na jinou webovou stránku.

Rozlišuje se více typů XSS. Prvním z nich je tzv. *Reflected XSS*. Kód útočníka se zde přenáší přes HTTP požadavek pro načtení webové stránky, nejčastěji v parametru URI, a je posléze interpretován prohlížečem. Útok se šíří přes odkazy obsahující škodlivý kód, které mohou být stejně jak u CSRF šířeny pomocí praktik sociálního inženýrství (Cross-site Scripting, 2018).

Další typem je tzv. *Stored XSS*, kde se škodlivý kód nachází přímo v úložišti na serveru. Nejčastěji se jedná o webové diskuzní fóra. Útočníkův kód se na stránku dostane prostřednictvím neochráněného vstupu, např. jako příspěvek pro fórum. Ovlivněná stránka pak působí škodu každému uživateli, který na ni v dobré víře přistoupí (LeBlanc a Messerschmidt, 2016).

Poslední, méně známý typ nese název *DOM-based XSS*. Útok je založen na útočníkově znalosti kódu webové stránky, na kterou chce zaútočit. Pokud stránka obsahuje kód stejného charakteru, jako ilustruje obrázek (Obr. 3.6), kde se první položka výčtového elementu *select* vytváří automaticky z hodnoty parametru *default*, má útočník prostor pro konstrukci URL adresy se škodlivým kódem. Princip se podobá typu *Reflected XSS* (DOM Based XSS, 2015).

Proti všem zmíněným typům XSS může vývojář zakročit ošetřením hodnot vstupů do aplikace - parametry URI, vstupní pole formulářů apod. Bežnou praxí je dovolit uživateli zadat pouze nezbytné znaky. Očekávali-li se například, že vstupní pole může

```
…

Select your language:

<select><script>

document.write("<OPTION
value=1>"+document.location.href.substring(document.location.href.indexOf("default=")+8)+"
</OPTION>");

document.write("<OPTION value=2>English</OPTION>");

</script></select>
…
```

Obr. 3.6 Příklad kódu náchylného na DOM-based XSS

obsahovat pouze číslo, neměly by být možné zadat písmeno. Další strategií je zakódování výstupu na stránku, která však dokáže útočníka maximálně zpomalit díky složitější konstrukci škodlivého kódu (Mueller, 2016). Na zmíněné praktiky proti XSS útoku byl kladen důraz i při vývoji výsledného systému této práce.

3.5 SQL Injection

Útok SQL Injection spočívá stejně jako XSS v integraci kódu útočníka do kódu aplikace. Jeho následky mohou být katastrofální, jelikož jazyk SQL pracuje na úrovni databáze a umožňuje operace pro modifikace dat, tabulek, schématu i celého databázového systému. Škodlivý kód se dostává do SQL dotazu při jeho dynamické konstrukci ze vstupních dat. Vykonání původní funkce je většinou potlačeno na úkor útoku (Mueller, 2016).

V raných dobách Internetu se s SQL Injection se potýkaly zejména aplikace postavené na jazyku PHP či ASP, jejichž rozhraní tehdy neobsahovaly automatické zábrany proti útoku. U moderních technologií dochází k nežádoucím manipulacím s SQL dotazy málokdy. Historie dala za vznik aplikačním rozhraním pro práci s databází, které mimo jiné i chrání systémy před hrozbami SQL Injection (SQL Injection, 2016).

3.6 Session Hijacking

Session Hijacking je popsán jako hrozba, kde se útočník dostává k cizí session ID. Jedná se o identifikátor relace, který server používá pro rozeznání uživatele a ověření platnosti autentizace. Podaří-li se útočníkovi získat tento identifikátor, může v systému působit pod identitou napadeného uživatele (Mueller, 2016).

Hrozba se týká webových aplikací, které přenáší session ID pomocí parametru v URL adrese. Jak již bylo avizováno, HTTP hlavička *Referer* obsahuje adresu předchozí stránky. Přesměruje-li se uživatel z ohroženého webu na web útočníka, dává mu tak

možnost zmocnit se jeho identity ve zdrojové aplikaci (Mueller, 2016). Adresa obsahující session ID může být získána i jinými způsoby, uživatel ji může např. sdílet s jinou osobou, neuvědomujíc si následků.

Vývojář může webovou aplikaci chránit tak, že session ID uchová v jiném úložišti na straně klienta. Nejčastěji se využívá souborů *cookies*. Základní protekce proti převzetí identity ve webové aplikaci je generování nepředvídatelného session ID (Mueller, 2016).

3.7 Bezpečnost v rámci životního cyklu systému

Dle Sommervilla (2013) by měl každý systém disponovat čtyřmi základními vlastnostmi. Systém by měl být *udržovatelný*, tedy připravený pro další možný rozvoj a měnící se požadavky zákazníků, *efektivní* z hlediska spotřeby zdrojů, *akceptovatelný* uživateli a také *spolehlivý*. Důležitost posledního atributu podporuje fakt, že chyby v systému mohou mít negativní vliv na velký počet lidí, způsobit značné peněžní škody či úniky informací.

Spolehlivost systému odráží jeho důvěryhodnost. Sommervile (2013) specifikoval spolehlivost jako kombinaci čtyř dimenzí:

- dostupnost a spolehlivost,
- bezpečnost,
- a zabezpečení.

Bezpečný systém by při svém selhání nikdy neměl ohrozit osoby ve svém prostředí či své okolí. Tato dimenze je důležitá zejména u řídících a monitorovacích systémech v letectví, systémů řízení procesů v továrnách, systémů podpory řízení v automobilech apod. Zabezpečením systému se myslí ochrana proti externím útokům, ať už náhodným či úmyslným. Tvůrci systému musí dbát na zajištění důvěryhodnosti, integrity a dostupnosti systému a jeho dat (Sommerville, 2013).

Ať už se systém vyvíjí modelem vodopádu, inkrementální či spirálovou metodou, vždy je součástí procesu testování výsledků implementace. Validace zabezpečení systému má však své úskalí. Požadavky na zabezpečení jsou totiž zpravidla popsány jako výčet toho, co systém nesmí dovolit, avšak postrádají žádané chování a popis reakce systému na hrozbu. Nechtěné chování aplikace se popisuje hůře než klasický funkční požadavek. Navíc jsou útočníci zpravidla inteligentní jedinci, kteří aktivně vyhledávají zranitelnosti systému tak, že se ho snaží používat v rozporu se standardním účelem. Využívají známých chyb vývojářů a bezpečnostních děr použitých technologií. Existují proto čtyři strategie pro testování bezpečnosti systémů:

1. testování na základě zkušeností,

2. testování *tiger teamem*,
3. testování za použití specializovaných nástrojů,
4. a testování vůči formální specifikaci (Sommervile, 2013).

Při testování bezpečnosti na základě zkušeností se sází na znalosti testerů. Mají-li například zkušenost s SQL Injection, snaží se simulovat napadení systému typickými pro tuto hrozbu. Pro správu a organizaci testování se zde často tvoří tzv. *security checklist*, tedy seznam možných hrozeb a výsledek validace (Sommervile, 2013).

Tiger Team je označení pro tým testerů, kteří disponují dostatečnou znalostí v oblasti zranitelnosti systémů. Členové se vybírají mimo řady vývojářů systému, aby bylo dosaženo jiné perspektivy na produkt. Tým má za úkol simulovat práci útočníků a snaží se narušit integritu systému a jeho dat.

Testování za použití specializovaných nástrojů spočívá v použití různých bezpečnostních nástrojů, jako např. validátor síly hesel (Sommerville, 2013). Existují i automatizované programy, které umí procházet systém a simulovat chování útočníka pomocí aplikace známých škodlivých praktik.

Poslední strategií je testování bezpečnosti systému vůči formální specifikaci hrozeb. Tento způsob validace však není příliš používaný (Sommerville, 2013).

Pro testování zranitelnosti výsledného systému byla vybrána strategie založená na zkušenosti s pomocí specializovaných bezpečnostních nástrojů. Na základě předchozích poznatků autora již byly v textu vyjmenovány a popsány hrozby typické pro webové aplikace, které byly během vývoje testovány manuálně i specializovanými programy.

II. PRAKTICKÁ ČÁST

4 Analýza systému

Cílem práce je vytvoření systému pro správu a automatizaci vydávání změn do prostředí systému Salesforce. Systém by měl sloužit vývojářům a administrátorům, kteří pracují na platformě Force.com a potřebují snadno přesouvat vyvinuté rozšíření systému z vývojového, přes testovací do produkčního prostředí.

Při implementaci rozšíření systému Salesforce dochází k modifikaci metadat. Jedná se o Visualforce stránky, třídy a triggery jazyka Apex, objekty v databázi, validační pravidla a další typy metadat. Dohromady Salesforce nabízí až 180 typů. Pro jejich přesun na jiné prostředí se v této práci používá pojem metadatová migrace.

Další modifikace pro vývoj rozšíření mohou být vykonány na úrovni záznamů v databázi systému Salesforce. Vznikají tak soubory dat, které je nutné zpracovat na jiném prostředí. Může se jednat o vložení nových záznamů či úpravu/smazání stávajících dat. V této práci jsou zmíněné činnosti vedeny jako datové migrační kroky.

Jelikož jsou databáze u jednotlivých prostředí Salesforce na sobě nezávislé, může při datových migracích dojít k situaci, kde není možné jednoduše upravit stávající záznamy na cílovém prostředí, jelikož záznamy ze zdrojového prostředí mají jiné identifikátory. Tato kolize lze řešit různými způsoby. Jedním z nich je úprava záznamů pomocí skriptů, které se spouští na cílovém prostředí. Skripty jsou psány v jazyce Apex a mají přímý přístup do databáze v prostředí. Umožňují tak výběr dat na základě různých kritérií a následnou úpravu. Pomocí skriptů lze automatizovaně měnit i některé nastavení v systému Salesforce.

Do některých konfigurací však nelze zasahovat ani pomocí migrace metadat či spuštění skriptu. Jedná se například o změny do funkce vyhledávání v systému Salesforce (Development Lifecycle Guide, 2016). Takové změny je vývojář nucen migrovat manuálně, tedy nastavit v rámci webového uživatelského prostředí.

Všechny typy migrací musí být řádně zdokumentovány. Výsledný systém tak nabízí možnost vytvořit tzv. migrační plány, které se skládají z migračních kroků v určeném pořadí. Lze tak podchytit závislost mezi jednotlivými úkony. V rámci migračního plánu je pak možné spustit samotné migrace dat či metadat, spouštět Apex skripty a označit manuální krok za splněný.

Výsledný systém vnáší do problematiky vydávání změn pro Salesforce CRM určitou míru automatizace a to zejména do migrace metadat. U nástrojů Change Set i Force.com Migration Tool je při přesunu metadat nutné získat balík komponent ze zdrojového prostředí a až pak spustit nahrávání do prostředí cílového, což v praxi zahrnuje hodně interakcí ze strany uživatele. Výsledný systém při provádění metadatové migrace spouští proces nahrávání automaticky, a to ihned po získání balíku komponent.

K úspoře času dochází i při migrací dat a Apex skriptů. Systém je navržen tak, aby umožnil spuštění těchto procesů jedním kliknutím na tlačítko a nenutil uživatele při vydávání změn pracovat s více softwarovými nástroji.

Implementaci systému předcházela důkladná analýza. Bylo nutné stanovit funkce systému, identifikovat aktéry a jejich role, nastavit procesy apod. Analýza se skládala z více fází, které byly provedeny v následujícím pořadí:

1. specifikace funkčních a mimofunkčních požadavků,
2. identifikace případů užití,
3. tvorba scénářů pro případy užití,
4. identifikace entit figurujících v systému,
5. a vytvoření diagramu tříd.

4.1 Specifikace požadavků

Na základě již zmíněných poznatků byly definovány funkční a mimofunkční požadavky na systém. Jejich tvorbu podpořily rozhovory s vývojáři, kteří pracují s platformou Force.com, a také osobní zkušenosti autora při vydávání změn.

Funkční požadavky popisují, co má systém provádět. Definují konkrétní poskytované funkce. Mimofunkční požadavky popisují omezení systému a jsou taktéž využity pro stanovení nároků na spolehlivost, dobu odezvy či vytíženost.

4.1.1 Funkční požadavky

Správa migračního plánu V systému bude možné vytvářet a upravovat migrační plány, které představují sérii kroků pro přesun funkcionality mezi prostředími systému Salesforce. Kroky mohou být manuální, např. povolení určité funkce v konfiguraci, či automatizované – přesun dat, přesun metadat, spuštění Apex skriptu. Pořadí kroků může být různé a je žádoucí, aby se jednotlivé typy kroků mohly opakovat. Bude například možné mít v rámci stejného migračního plánu dva kroky na migraci metadat, mezi kterými je nutné provést manuální krok.

Spouštění automatizovaných migračních kroků V systému bude možné spustit automatizované migrační kroky. V rámci vytváření automatizovaného kroku bude možné přiložit veškeré potřebné soubory pro spuštění procesu migrace.

Správa metadatových manifestů V systému bude možné vytvářet, upravovat a mazat seznamy souborů pro migraci – tzv. metadatové manifesty. Místo toho, aby uživatel vytvářel a uchovával manifest na svém lokálním úložišti, systém nabídne nástroj, který se připojí k prostředí systému Salesforce, nabídne dostupná metadata a umožní vytvoření vlastního seznamu, který se uloží na serveru a bude k dispozici při tvorbě automatizovaného migračního kroku pro přenos metadat.

Správa přístupů k prostředím systému Salesforce V systému bude možné evidovat, k jakým prostředím systému Salesforce má uživatel přístup. Účelem je usnadnit výběr zdrojového a cílového prostředí v migračním kroku a také urychlit proces autentizace.

Registrace a přihlášení Systém bude uživatelům nabízet možnost si vytvořit přístupové údaje do systému. Při registraci bude vyžadována emailová adresa uživatele, která zároveň poslouží jako uživatelské jméno, a dostatečně silné přístupové heslo. Autentizace uživatele bude probíhat na základě poskytnutého uživatelského jména a hesla.

4.1.2 Mimofunkční požadavky

Responzivní design Uživatelské prostředí systému se bude přizpůsobovat velikosti obrazovky zařízení, na kterém bude provozováno. Výsledkem bude pohodlné používání systému na klasickém počítači, tabletu i mobilním telefonu.

Ukládání hesel k prostředím systému Salesforce Z bezpečnostních důvodů nebudou ukládány přístupová hesla k prostředím systému Salesforce do databáze systému. Systém bude pracovat s dočasnými přístupovými tokeny.

Vystopovatelnost detailů manipulace s daty V systému bude možné provést audit manipulací s daty. Pro objekty z doménového modelu bude evidováno, kdo a kdy vytvořil, naposledy upravil či smazal konkrétní záznam.

4.2 Případy užití

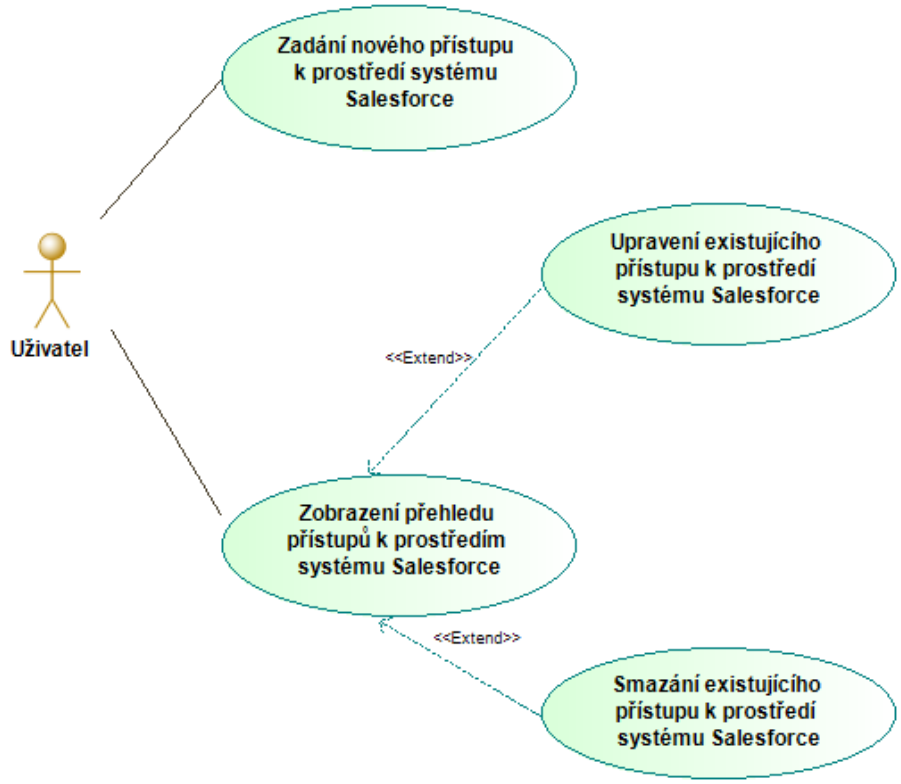
Na základě funkčních a mimofunkčních požadavků byly identifikovány aktéři figurující v systému:

- uživatel,
- zdrojové prostředí systému Salesforce,
- a cílové prostředí systému Salesforce.

Dále vznikly případy užití, které rozebírají vyžadované funkce do větších podrobností. Jsou seskupeny do funkčních celků a vizualizovány do diagramů pomocí standardů jazyka UML. Ke každému případu užití taktéž náleží scénář, který popisuje interakce mezi uživatelem a systémem.

4.2.1 Správa přístupů k prostředím systému Salesforce

První funkční celek se zabývá možnostmi správy přístupů k prostředím systému Salesforce. Případy užití, které seskupuje, jsou prerekvizitami pro další činnosti týkající se tvorby migračních kroků. Grafické znázornění celku představuje obrázek (Obr. 4.1).



Obr. 4.1 Diagram případů užití pro správu přístupů k prostředím systému Salesforce

Zadání nového přístupu k prostředí systému Salesforce

1. Případ užití začíná, když uživatel zvolí v nabídce položku pro vytvoření přístupu k prostředí systému Salesforce.
2. Systém zobrazí formulář pro vytvoření přístupu.
3. Uživatel do formuláře vyplní uživatelské jméno k prostředí, typ prostředí, případně i vlastní URL adresu prostředí.

4. Uživatel zvolí položku po uložení přístupu.
5. Systém uloží přístup k prostředí.
6. Případ užití končí.

Zobrazení přehledu přístupů k prostředí systému Salesforce

1. Případ užití začíná, když uživatel zvolí v nabídce položku pro zobrazení přehledu přístupů k prostředí systému Salesforce.
   (a) ALT Případ užití začíná, když uživatel dokončí vytváření přístupu k prostředí.
   (b) ALT Případ užití začíná, když uživatel dokončí úpravu přístupu k prostředí.
2. Systém zobrazí přehled přístupů.
3. IF uživatel zvolí na přístupu k prostředí položku úpravy
   (a) EXTEND Upravení existujícího přístupu k prostředí systému Salesforce.
4. IF uživatel zvolí na přístupu k prostředí položku smazání
   (a) EXTEND Smazání existujícího přístupu k prostředí systému Salesforce.
5. Případ užití končí.

Upravení existujícího přístupu k prostředí systému Salesforce

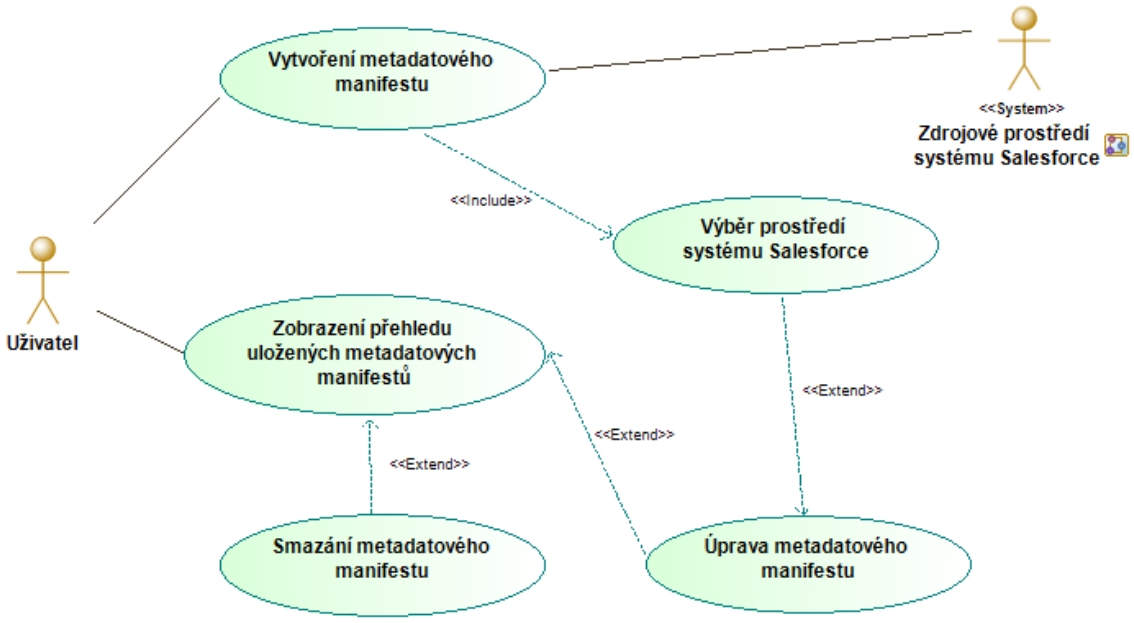
1. Případ užití začíná, když uživatel v přehledu přístupů k prostředím zvolí možnost úpravy určeného přístupu.
2. Systém zobrazí formulář, do kterého zobrazí již uložené informace o přístupu.
3. Uživatel upraví data ve formuláři.
4. Uživatel zvolí položku po uložení přístupu.
5. Systém uloží přístup k prostředí.
6. Případ užití končí.

Smazání existujícího přístupu k prostředí systému Salesforce

1. Případ užití začíná, když uživatel v přehledu přístupů k prostředím zvolí možnost smazání určeného přístupu.
2. Systém zobrazí dialog pro potvrzení volby smazání přístupu.
3. Pokud uživatel volbu potvrdí, systém vymaže přístup k prostředí z databáze.
4. Případ užití končí.

4.2.2 Správa metadatových manifestů

Další funkční celek obsahuje případy užití, které se zabývají tvorbou a úpravou metadatových manifestů - seznamů zdrojových souborů pro migraci. Mimo uživatele se případů užití účastní i prostředí systému Salesforce, které slouží jako zdroj seznamu všech metadat a ze kterého je vybírána podmnožina metadat pro manifest. Diagram funkčního celku znázorňuje obrázek (Obr. 4.2).



Obr. 4.2 Diagram případů užití pro správu metadatových manifestů

Vytvoření metadatového manifestu

1. Případ užití začíná, když uživatel zvolí v nabídce položku pro vytvoření metadatového manifestu.
2. Systém zobrazí formulář pro vytvoření nového manifestu.

3. Uživatel do formuláře vyplní základní informace o manifestu.
    (a) INCLUDE Výběr prostředí systému Salesforce.
4. Systém poskytne pole pro zadání přístupového hesla do vybraného prostředí.
5. Uživatel vyplní heslo a potrdí volbu.
6. Systém z vybraného prostředí získá seznam dostupných metadat a zobrazí je do formuláře tvorby manifestu.
7. Uživatel zvolí, které metadata budou součástí manifestu.
8. Uživatel zvolí položku pro uložení manifestu.
9. Systém provede uložení manifestu.
10. Případ užití končí.

Zobrazení přehledu uložených metadatových manifestů

1. Případ užití začíná, když uživatel zvolí v nabídce položku pro zobrazení přehledu svých metadatových manifestů.
    (a) ALT Případ užití začíná, když uživatel dokončí vytvoření nového metadatoéhé manifestu.
    (b) ALT Případ užití začíná, když uživatel dokončí úpravu existujícího metadatového manifestu.
    (c) ALT Případ užití začíná, když uživatel smaže metadatový manifest.
2. Systém zobrazí přehled existujících metadatových manifestů uživatele.
3. IF uživatel zvolí na detailu metadatového manifestu položku úpravy
    (a) EXTEND Úprava metadatového manifestu.
4. IF uživatel zvolí na detailu metadatového manifestu položku smazání
    (a) EXTEND Smazání metadatového manifestu.
5. Případ užití končí.

Úprava metadatového manifestu

1. Případ užití začíná, když uživatel v přehledu svých metadatových manifestů zvolí možnost úpravy manifestu.
2. Systém zobrazí formulář a vyplní ho daty z upravovaného manifestu.
3. Uživatel upraví informace o manifestu.
4. IF uživatel zvolí možnost výběru metadat z jiného prostředí systému Salesforce
   (a) EXTEND Výběr prostředí systému Salesforce.
5. Systém poskytne pole pro zadání přístupového hesla do vybraného prostředí.
6. Uživatel vyplní heslo a potrdí volbu.
7. Systém z vybraného prostředí získá seznam dostupných metadat a zobrazí je do formuláře tvorby manifestu.
8. Uživatel zvolí, která metadata budou součástí manifestu.
9. Uživatel zvolí volbu pro uložení manifestu.
10. Systém provede uložení manifestu.
11. Případ užití končí.

Výběr prostředí systému Salesforce

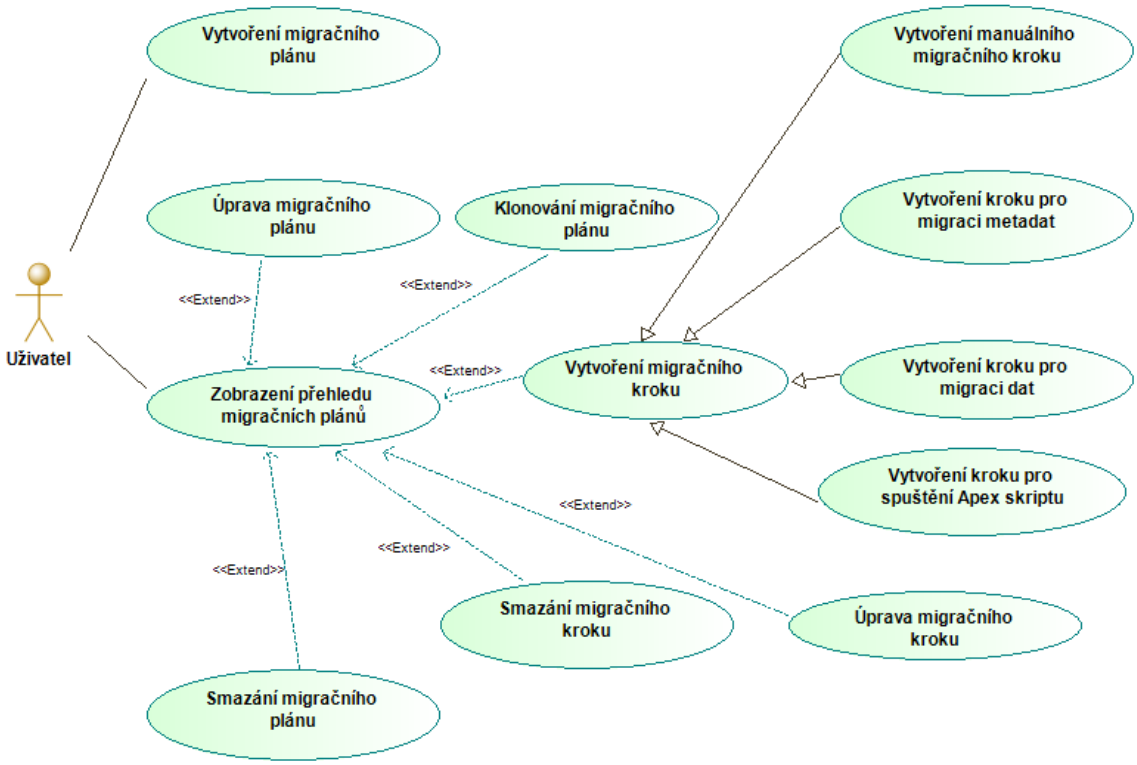
1. Případ užití začíná, když uživatel zvolí položku pro výběr prostředí systému Salesforce při vytváření manifestu.
   (a) ALT Případ užití začíná, když uživatel zvolí položku pro výběr prostředí systému Salesforce při úpravě manifestu.
2. Systém zobrazí přehled uložených přístupů do prostředí systému Salesforce.
3. Uživatel z přehledu vybere prostředí, ze které chce zobrazit seznam metadat.
4. Systém si volbu uloží pro následující operace.
5. Případ užití končí.

Smazání metadatového manifestu

1. Případ užití začíná, když uživatel v přehledu svých metadatových manifestů zvolí možnost smazání manifestu.
2. Systém zobrazí dialog pro potvrzení volby smazání manifestu.
3. Pokud uživatel volbu potvrdí, systém vymaže manifest z databáze.
4. Případ užití končí.

4.2.3 Správa migračního plánu

Třetí funkční celek obsahuje případy užití, které se týkají vzniku a úprav migračního plánu. Plán se skládá z migračních kroků, které je rovněž potřeba vytvořit, případně upravit a smazat. Případy užití v rámci diagramu znázorňuje obrázek (Obr. 4.3).



Obr. 4.3 Diagram případů užití pro správu migračního plánu

Vytvoření migračního plánu

1. Případ užití začíná, když uživatel zvolí v nabídce položku pro vytvoření migračního plánu.

2. Systém zobrazí formulář pro vytvoření nového migračního plánu s možností specifikovat veškeré potřebné vlastnosti plánu.
3. Uživatel vyplní formulář a potvrdí tvorbu plánu tlačítkem.
4. Systém vytvoří nový plán.
5. Systém uživatele přesměruje na náhled nově vytvořeného migračního plánu, který zatím neobsahuje žádné migrační kroky.
6. Případ užití končí.

Zobrazení přehledu migračních plánů

1. Případ užití začíná, když uživatel zvolí v nabídce položku pro zobrazení přehledu migračních plánů.
2. Systém zobrazí přehled uložených migračních plánů.
3. IF uživatel zvolí položku pro úpravu migračního plánu
   (a) EXTEND Úprava migračního plánu.
4. IF uživatel zvolí položku pro klonování migračního plánu
   (a) EXTEND Klonování migračního plánu.
5. IF uživatel zvolí položku pro smazání migračního plánu
   (a) EXTEND Smazání migračního plánu.
6. IF uživatel zvolí položku pro vytvoření migračního kroku
   (a) EXTEND Vytvoření migračního kroku.
7. IF uživatel u migračního kroku zvolí položku pro úpravu kroku
   (a) EXTEND Úprava migračního kroku.
8. IF uživatel u migračního kroku zvolí položku pro smazání kroku
   (a) EXTEND Smazání migračního kroku.
9. Případ užití končí.

Úprava migračního plánu

1. Případ užití začíná, když uživatel v přehledu migračních plánů zvolí položku pro úpravu určitého plánu.
2. Systém zobrazí formulář a vyplní ho daty z upravovaného plánu.
3. Uživatel upraví informace o plánu.
4. Uživatel zvolí položku pro uložení plánu.
5. Systém uloží upravený plán.
6. Případ užití končí.

Klonování migračního plánu

1. Případ užití začíná, když uživatel v přehledu migračních plánů zvolí položku pro klonování určitého plánu.
2. Systém zobrazí formulář pro vytvoření nového plánu a vyplní ho daty z klonovaného plánu.
3. Je-li potřeba, uživatel upraví informace o plánu.
4. Uživatel zvolí položku pro uložení plánu.
5. Systém uloží plán jako nově vytvořený a naklonuje automaticky i migrační kroky, které obsahuje původní plán.
6. Případ užití končí.

Smazání migračního plánu

1. Případ užití začíná, když uživatel v přehledu migračních plánů zvolí položku pro smazání určitého plánu.
2. Systém zobrazí dialog pro potvrzení volby smazání plánu.
3. Pokud uživatel volbu potvrdí, systém vymaže plán z databáze.
4. Případ užití končí.

Vytvoření migračního kroku Případ užití týkající se tvorby migračního kroku je abstraktní, jelikož systém rozlišuje migrační krok manuální či automatizovaný. Dělí se tak na více případů užití:

- vytvoření manuálního migračního kroku,
- vytvoření kroku pro migraci metadat,
- vytvoření kroku pro migraci dat a
- vytvoření kroku pro spuštění Apex skriptu.

Vytvoření manuálního migračního kroku

1. Případ užití začíná, když uživatel zvolí v migračním plánu položku pro vytvoření nového manuálního migračního kroku.
2. Systém zobrazí formulář pro nový manuální migrační krok. Předpokládá se jediné textové pole pro popis kroku.
3. Uživatel vyplní formulář.
4. Uživatel zvolí položku pro uložení migračního kroku.
5. Systém uloží migrační krok.
6. Případ užití končí.

Vytvoření kroku pro migraci metadat

1. Případ užití začíná, když uživatel zvolí v migračním plánu položku pro vytvoření nového kroku pro migraci metadat.
2. Systém zobrazí formulář pro nový migrační krok.
3. Uživatel vyplní formulář a označí metadatové manifesty pro migraci.
4. Uživatel zvolí položku pro uložení migračního kroku.
5. Systém uloží krok pro migraci metadat do databáze.
6. Případ užití končí.

Vytvoření kroku pro migraci dat

1. Případ užití začíná, když uživatel zvolí v migračním plánu položku pro vytvoření nového kroku pro migraci dat.
2. Systém zobrazí formulář pro nový migrační krok.
3. Uživatel vyplní formulář a vloží potřebné soubory – cílový objekt v databázi Salesforce, soubor s daty a DML operaci.
4. Uživatel zvolí položku pro uložení migračního kroku.
5. Systém uloží krok pro migraci dat do databáze.
6. Případ užití končí.

Vytvoření kroku pro spuštění Apex skriptu

1. Případ užití začíná, když uživatel zvolí v migračním plánu položku pro vytvoření nového kroku pro spuštění Apex skriptu.
2. Systém zobrazí formulář pro nový migrační krok.
3. Uživatel vyplní formulář a vloží soubor obsahující skript.
4. Uživatel zvolí položku pro uložení migračního kroku.
5. Systém uloží migrační krok.
6. Případ užití končí.

Úprava migračního kroku

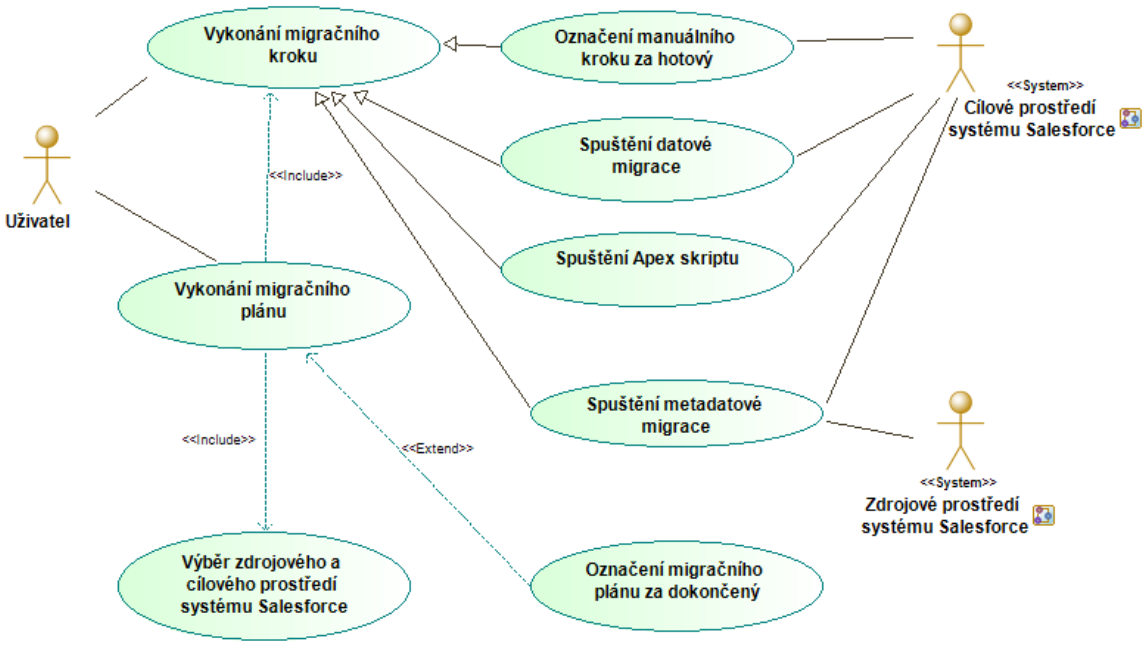
1. Případ užití začíná, když uživatel zvolí v migračním plánu položku pro úpravu migračního kroku.
2. Systém zobrazí formulář, do kterého zobrazí již uložené informace o migračním kroku.
3. Uživatel upraví data ve formuláři.
4. Uživatel zvolí položku po uložení migračního kroku.
5. Systém uloží migrační krok.
6. Případ užití končí.

Smazání migračního kroku

1. Případ užití začíná, když uživatel zvolí v migračním plánu položku pro smazání migračního kroku.
2. Systém zobrazí dialog pro potvrzení volby smazání migračního kroku.
3. Pokud uživatel volbu potvrdí, systém vymaže migrační krok z databáze.
4. Pokud je součástí migračního kroku soubor pro migraci dat či spuštění Apex skriptu, i tyto soubory systém vymaže z databáze.
5. Případ užití končí.

4.2.4 Realizace migračního plánu

Další funkční celek zastřešuje případy užití, které se týkají samotných migrací. Migrační plán, jehož tvorba byla popsána v přechozím celku, zde přechází do fáze realizace. Případů užití se mimo uživatele účastní i zdrojové prostředí a cílové prostředí systému Salesforce. Diagram představuje obrázek (Obr. 4.4).



Obr. 4.4 Diagram případů užití pro realizaci migračního plánu

Vykonání migračního plánu

1. Případ užití začíná, když se uživatel rozhodne vykonávat kroky migračního plánu.

   (a) INCLUDE Výběr zdrojového a cílového prostředí systému Salesforce.
   (b) INCLUDE Vykonání migračního kroku.

2. IF migrační plán je dokončen
   (a) EXTEND Označení migračního plánu za dokončený.
3. Případ užití končí.

Výběr zdrojového a cílového prostředí systému Salesforce

1. Případ užití začíná, když se uživatel rozhodne vykonat migrační plán.
2. IF migrační plán obsahuje krok na migraci metadat
   (a) Systém zobrazí seznam uživatelových přístupů do prostředí systému Salesforce.
   (b) Uživatel vybere zdrojové prostředí.
   (c) Systém zobrazí formulář pro zadání hesla a bezpečnostního tokenu pro vybrané prostředí.
   (d) Uživatel vyplní heslo, případně i bezpečnostní token a potvrdí vstup.
   (e) Systém provede autentizaci a autorizaci vůči zdrojovému prostředí systému Salesforce.
   (f) Systém uloží přístupový token k zdrojovému prostředí.
3. Systém zobrazí seznam uživatelových přístupů do prostředí systému Salesforce.
4. Uživatel vybere cílové prostředí.
5. Systém zobrazí formulář pro zadání hesla a bezpečnostního tokenu pro vybrané prostředí.
6. Uživatel vyplní heslo, případně i bezpečnostní token a potvrdí vstup.
7. Systém provede autentizaci a autorizaci vůči cílovému prostředí systému Salesforce.
8. Systém uloží přístupový token k cílovému prostředí.
9. Případ užití končí.

Vykonání migračního kroku Případ užití pro realizaci migračního kroku je abstraktní, jelikož se musí rozlišovat vykonání manuálního a automatizovaného migrační kroku. Manuální krok je popsán v případu užití Označení manuálního kroku za hotový. Automatické migrační kroky jsou pokryty v případech Spuštění metadatové migrace, Spuštění datové migrace a Spuštění Apex skriptu.

Označení manuálního kroku za hotový

1. Případ užití začíná, když se uživatel rozhodne realizovat manuální migrační krok.
2. Uživatel vykoná manuální krok v cílovém prostředí systému Salesforce.
3. Uživatel v systému označí migrační krok za hotový.
4. Systém uloží nový stav migračního kroku.
5. Případ užití končí.

Spuštění metadatové migrace

1. Případ užití začíná, když uživatel v migračním plánu zvolí položku pro spuštění migrace metadat.
2. Systém použije přiložený manifest a přístupový token k zdrojovému prostředí systému Salesforce, aby získal metadata ze zdrojového prostředí.
3. Systém uloží získaná metadata.
4. Systém použije přístupový token k cílovému prostředí systému Salesforce a spustí nasazení získaných metadat na cílového prostředí.
5. Systém získá výsledek nasazení.
6. Systém zobrazí výsledek nasazení.
7. Pokud bylo nasazení úspěšné, systém označí migrační krok za hotový.
8. Systém vymaže získaná metadata ze své paměti.
9. Případ užití končí.

Spuštění datové migrace

1. Případ užití začíná, když uživatel v migračním plánu zvolí položku pro spuštění migrace dat.
2. Systém použije přiložený CSV soubor s daty a přístupový token k cílovému prostředí systému Salesforce, aby na prostředí spustil nasazení dat.
3. Systém získá výsledek nasazení.
4. Systém zobrazí výsledek nasazení.
5. Pokud bylo nasazení úspěšné, systém označí migrační krok za hotový.
6. Případ užití končí.

Spuštění Apex skriptu

1. Případ užití začíná, když uživatel v migračním plánu zvolí položku pro spuštění Apex skriptu.

1. Systém použije přiložený textový soubor obsahující skript a přístupový token k cílovému prostředí systému Salesforce, aby na prostředí spustil vykonání skriptu.

1. Systém získá výsledek vykonání skriptu.
2. Systém zobrazí výsledek vykonání.
3. Pokud bylo vykonání úspěšné, systém označí migrační krok za hotový.
4. Případ užití končí.

Označení migračního plánu za dokončený

1. Případ užití začíná, když uživatel v migračním plánu zvolí položku pro označení plánu za dokončený.
2. Systém zkontroluje stav všech migračních kroků. Pokud nejsou všechny migrační kroky označeny jako hotové, upozorní uživatele o této situaci.
3. Systém uloží nový stav migračního plánu.
4. Případ užití končí.

4.3 Doménový model

V rámci analýzy byl sestaven i tzv. doménový model, který znázorňuje entity figurující v systému a vztahy mezi nimi. Identifikace entit proběhla na základě rozboru požadavků a případů užití. Doménový model je prezentován jako UML diagram tříd, který se kvůli své velikosti nachází v příloze (P 1) této práce. Jelikož systém používá funkce technologie Entity Framework za postupu *code-first*, model slouží i jako šablona pro generování tabulek v databázi. Následující text popisuje třídy z diagramu a jejich účel v systému.

4.3.1 Třída ApplicationUser

Uživatel výsledného systému je reprezentován třídou *ApplicationUser*. Třída dědí vlastnosti a operace třídy *IdentityUser* z již popsané technologie ASP.NET Core Identity. Díky tomu může využívat aplikační rozhraní pro autentizaci a autorizaci.

4.3.2 Třída AuditableEntity

Na základě mimofunkčního požadavku pro vystopovatelnost modifikací záznamů a požadavku pro *soft-delete* funkcionalitu byla vytvořena třída *AuditableEntity*. Třída má atributy, které odkazují na tvůrce záznamu (*CreatedBy*) a posledního uživatele, jenž záznam upravil či vymazal (*LastModifiedBy*). Pro každou datovou operaci je uložena i časová známka (*CreatedDate*, *LastModifiedDate*, *DeletedDate*). Příznak smazaného záznamu představuje atribut *IsDeleted*.

4.3.3 Třída SfOrgCredentials

Třída *SfOrgCredentials* reprezentuje přístupové údaje k organizaci systému Salesforce. Hodnota atributu *Username* je uživatelské jméno, heslo se na základě mimofunkčních požadavků neukládá. Atribut *OrgType* odlišuje, zda se přístupové údaje vztahují k produkčnímu prostředí či prostředí sandbox. Funguje-li prostředí na vlastní adrese URL, je uložena v atributu *CustomUrl*. Vlastnost *LoginUrl* pomáhá snadnému získání adresy pro autentizaci a je vyhodnocena na základě atributů *OrgType* a *CustomUrl*.

4.3.4 Třída SfMetadataManifest

Třída *SfMetadataManifest* představuje manifest pro metadatovou migraci. Atribut *Resources* obsahuje samotný seznam komponent pro přenos. Data v atributu reprezentuje slovník, kde je klíčem metadatový typ (např. *ApexPage*, *ApexClass*) a hodnotou množina komponent - názvy stránek, objektů apod. Atribut *ApiVersion* značí, jaká se použije verze služby Salesforce Metadata API. Některé typy metadat jsou dostupné

pro přesun až od určitých verzí aplikačního rozhraní. Pro rozlišení manifestů v rámci uživatelského prostředí slouží atribut *Name*.

4.3.5 Třída MigrationStep a její potomci

Třída *MigrationStep* představuje obecný migrační krok. Má atributy společné pro všechny typy migrací - pořadí kroku v plánu (*Order*), stav vykonávání kroku (*Status*) a odkaz na migrační plán (*ParentMigrationPlan*). Třída je abstraktní, jelikož v systému není potřeba vytvářet obecný krok, nýbrž krok pro konkrétní typ migrace. Slouží tak pouze jako společný základ atributů a operací pro její potomky.

Pro správu manuálních kroků vznikla třída *ManualStep*. Přidává atribut *Instruction* k uložení instrukcí pro vykonání kroku. Text v atributu může být strukturován pomocí značek jazyka HTML.

Třída *SfMetadataMigrationStep* představuje krok pro migraci metadat. Pro přesun může být využito více metadatových manifestů, čímž vzniká vazba M:N mezi třídami *SfMetadataMigrationStep* a *SfMetadataManifest*, která byla vyřešena přidáním asociační třídy *SfMetadataManifestStepAssignment* evidující přiřazení manifestu k migračnímu kroku.

Datový migrační krok reprezentuje třída *SfDataMigrationStep*. K atributům rodiče přidává informaci o typu DML operace (*DmlOperation*), jméno cílového objektu v databázi systému Salesforce (*ObjectApiName*) a samotný objekt s daty (*DataFile*), který je uložen jako pole bajtů.

Poslední typ kroku se týká spouštění Apex skriptů. Je realizován pomocí třídy *SfApexScriptRunStep*. Přidává atribut *ScriptFile*, který obsahuje soubor se skriptem. Po vzoru kroku pro datovou migraci je soubor uložen jako pole bajtů.

4.3.6 Třída MigrationPlan

Migrační kroky jsou součástí migračního plánu, který reprezentuje třída *MigrationPlan*. Dle požadavků třída eviduje stav vykonávání plánu (*Status*), krátký popis (*Description*) a datum plánovaného spuštění migrace (*ReleaseDate*). Plán lze pojmenovat pomocí atributu *Name*.

5 Implementace systému

Analýzu systému následovala jeho implementace. Jak již bylo uvedeno, výsledný systém reprezentuje webová aplikace implementovaná pomocí technologie ASP.NET Core 2 s daty uloženými v databázovém systému PostgreSQL 10. Následující text popisuje nejvýznamnější funkcionality systému a jak jich bylo dosaženo. Zdrojový kód aplikace se nachází v příloze této práce.

5.1 Architektura aplikace

Za použití praktik architektury MVC byl kód aplikace rozdělen do několika vrstev. Technologie ASP.NET Core využívá osvědčeného postupu, kde každé vrstvě náleží složka se zdrojovými soubory. Do výsledného systému byly začleněny další vrstvy a složky se specifickým účelem. Tabulka (Tab. 5.1) nabízí názvy a účely důležitých částí systému.

Tab. 5.1 Složky webové aplikace obsahující zdrojový kód

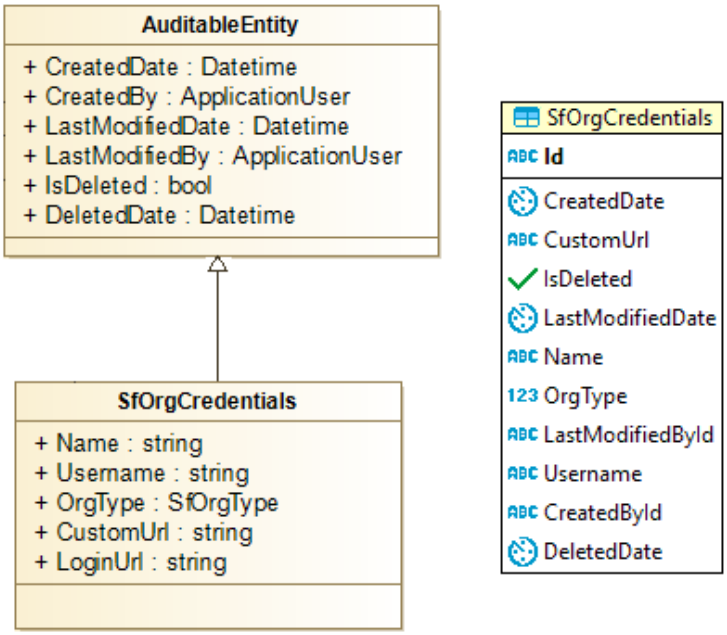
Název složky	**Charakter obsažených souborů**
Authorization	třídy pro vyhodnocení oprávnění k určitým akcím
Controllers	třídy zpracovávající interakce uživatele
Data	třídy nastavující parametry pro Entity Framework
Models	třídy doménového modelu, pomocné struktury
Services	třídy řešící určitou komplexní logiku systému
Views	vizuální část aplikace, webové šablony

5.2 Komunikace s databází

Na databázovém systému PostgreSQL byla vytvořena databáze dedikovaná pro webovou aplikaci. Systém komunikuje s databází pomocí technologie Entity Framework. Parametry pro připojení (IP adresa, port, přístupové jméno a heslo) se nachází v konfiguračním souboru výsledné aplikace. Přístupové údaje využívá třída *ApplicationDbContext*, která byla vytvořena za účelem zprostředkování komunikace mezi aplikací a databázovým systémem. Potřebné funkce dědí z třídy *DbContext* aplikačního rozhraní Entity Framework.

Tabulky databázového schématu jsou generovány automaticky pomocí nástroje .NET Core CLI. Jako předloha slouží třídy představené v doménovém modelu. Jedná se o již avizovaný postup *code-first*. Nástroj automaticky rozpoznal vztahy mezi objeky včetně agregací a dědičností. Příkladem je situace na obrázku (Obr. 5.1), kde se na levé straně nachází třída *SfOrgCredentials* a na pravé straně stejnojmenná tabulka. Nástroj .NET Core CLI vzal v úvahu dědičnost mezi třídami *SfOrgCredentials* a *AuditableEntity*

a v automaticky generované tabulce vytvořil převzaté atributy. Tento postup řešení dědičnosti při mapování modelu do databáze se nazývá *Table Per Concrete Type*. Postup byl aplikován i na ostatní potomky třídy *AuditableEntity*.



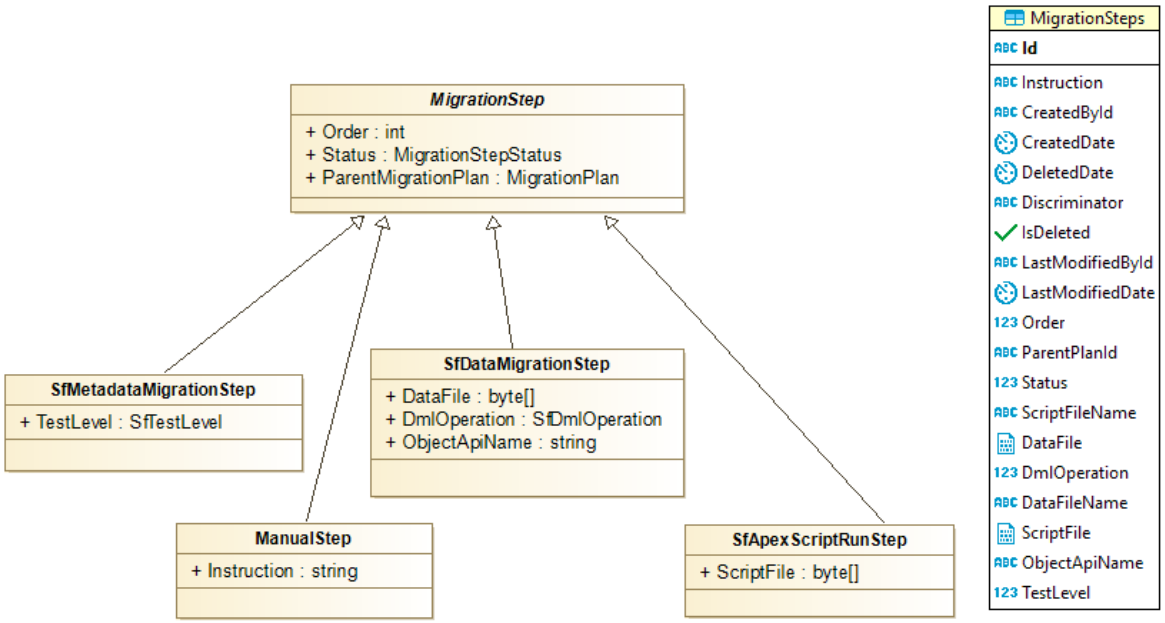
Obr. 5.1 Mapování třídy SfOrgCredentials na tabulku v databázi

Alternativní přístup k dědičnosti v rámci relační databáze spočívá v reprezentaci rodičů a potomků jedinou tabulkou, která obsahuje atributy všech zúčastněných tříd. Jedná se o strategii *Table Per Hierarchy* a v rámci výsledné aplikace byla aplikována na třídy reprezentující migrační kroky. Situaci ilustuje obrázek (Obr. 5.2). Třída *MigrationStep* a její potomci jsou v databázi zastoupeny jedinou tabulkou *SfMigrationSteps*. Implicitní atribut *Discriminator* uchovává datový typ potomka (např. *SfMetadataMigrationStep*) a umožňuje v aplikaci pracovat s vlastnostmi a operacemi konkrétního migračního kroku.

5.3 Autentizace vůči systému Salesforce

Aby bylo možné používat funkce Salesforce Metadata API a dalších zapojených webových služeb, je nutné nejprve ověřit totožnost uživatele vůči prostředí systému Salesforce. Výsledný systém k autentizaci využívá funkci *login()* z aplikačního rozhraní Salesforce SOAP API. HTTP požadavek obsahuje uživatelské jméno a heslo, v odpovědi se vrací přístupový token a doba jeho expirace.

Pro obsloužení webové služby vzniklo v systému rozhraní *ISfLoginService*, které nabízí metodu *LoginAsync*. Rozhraní realizuje třída *SfLoginService*. Její kód zahrnuje



Obr. 5.2 Mapování tříd migračních kroků na tabulku v databázi

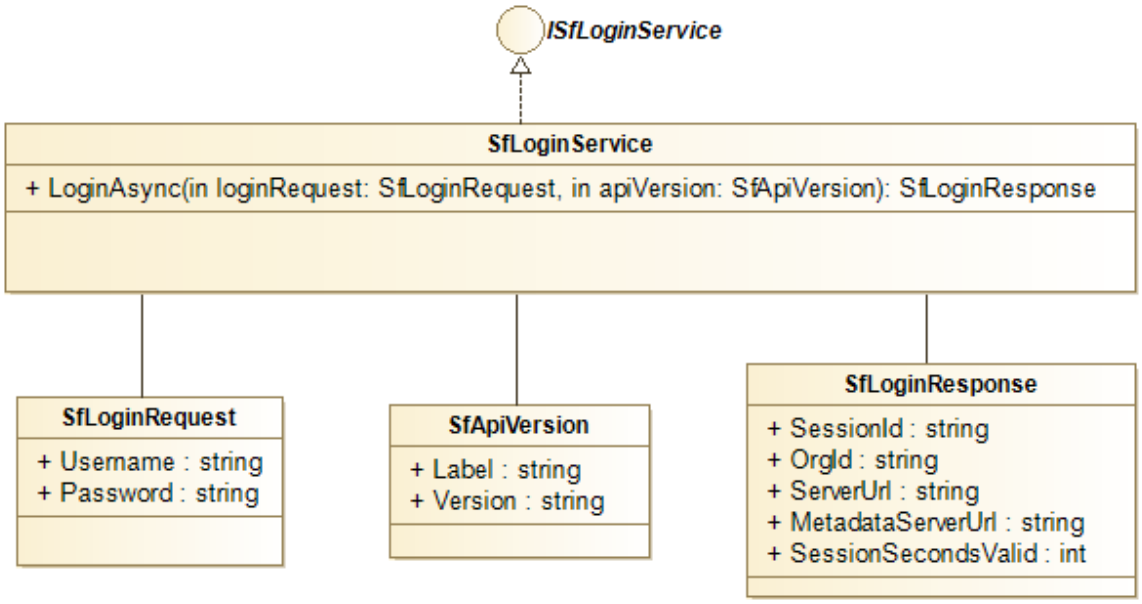
logiku pro autentizaci vůči systému Salesforce. Parametrem metody *LoginAsync* je instance třídy *SfLoginRequest* z vrstvy model, která odpovídá žádané struktuře požadavku pro akci *login()*. Dále se předává URL adresa prostředí a požadovaná verze aplikačního rozhraní. Metoda vrací instanci třídy *SfLoginResponse*, jež je deserializována z odpovědi na HTTP požadavek, a obsahuje všechny potřebné atributy pro další interakce v systému. Popsanou strukturu tříd znázorňuje UML diagram na obrázku (Obr. 5.3).

Rozhraní se stalo součástí tříd z vrstvy *Controller*, jejichž logika vyžaduje autentizaci vůči systému Salesforce. Taková třída obsahuje atribut datového typu *ISfLoginService* (rozhraní), do kterého je vložena konkrétní instance třídy *SfLoginService* (realizace rozhraní) až za běhu programu. Využívá se praktik návrhového vzoru Dependency Injection.

5.4 Migrace metadat

Pro migraci metadat se používá již avizovaná webová služba Salesforce Metadata API. Po vzoru autentizace vůči systému Salesforce je k volání funkcí služby vytvořena třída *SfMetadataService*, která realizuje rozhraní *ISfMetadataService* a která je vkládána do tříd vrstvy *Controller* pomocí vzoru Dependency Injection. Třída implementuje metody:

- *RetrieveAsync* pro zahájení procesu tvorby balíku komponent ze zdrojového prostředí,



Obr. 5.3 Struktura tříd umožňující autentizaci vůči systému Salesforce

- *CheckRetrieveStatusAsync* pro kontrolu stavu procesu tvorby a případné získání obsahu balíku,
- *DeployAsync* pro zahájení procesu nahrávání získaného balíku na cílové prostředí,
- a *CheckDeployStatusAsync* pro získání stavu procesu nahrávání.

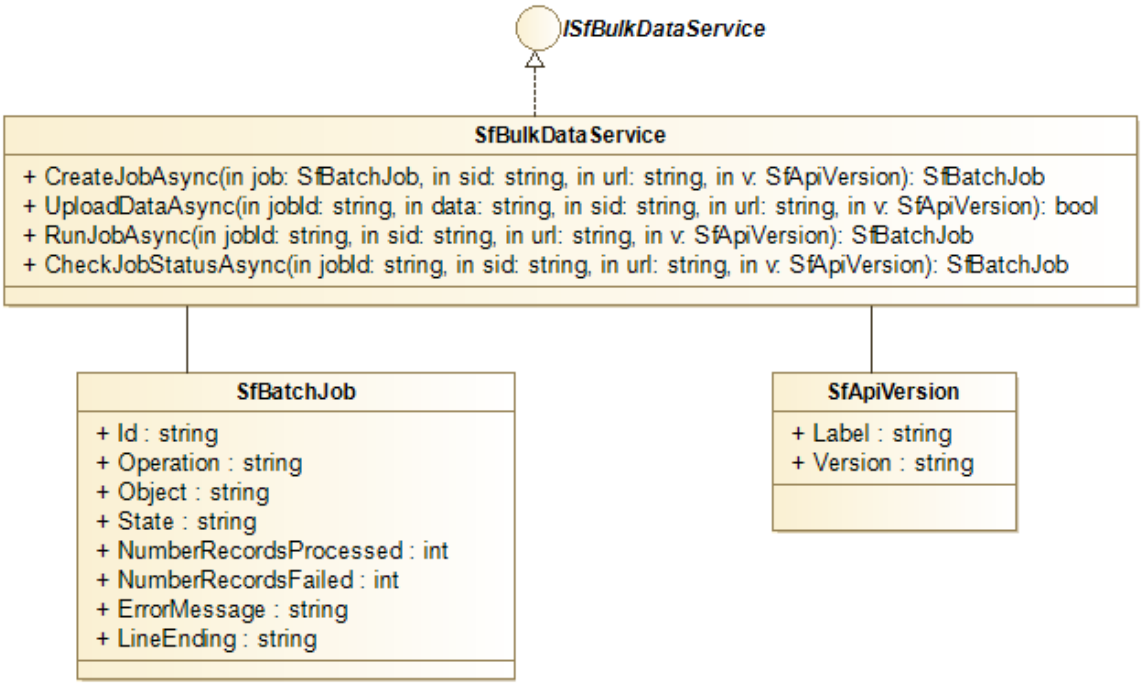
Každá z metod vyžaduje v parametru instanci třídy, která odpovídá předepsané podobě HTTP požadavku z dokumentace Salesforce Metadata API. Třídu *SfMetadataService* a související struktury znázorňuje UML diagram na obrázku v příloze (P 2). Důležitou součástí požadavku je přístupový token (*SessionId*), který lze získat pomocí služby pro autentizaci vůči systému Salesforce.

Přesun metadat ze zdrojového prostředí na cílové vyžaduje použití všech výše popsaných metod. Metoda *RetrieveAsync* zahajuje proces získávání balíku metadat. Poté je potřeba v rozumném časovém intervalu kontrolovat stav procesu pomocí metody *CheckRetrieveStatusAsync*, která vrací stav procesu. V případě, že proces již skončil a nedošlo k chybovému stavu, je součástí výsledku i zakódovaný obsah balíku metadat. Na stejném asynchronním principu funguje nahrávání balíku metadat na cílové prostředí pomocí metody *DeployAsync* a kontrola pravidelným voláním *CheckDeployStatusAsync*. Životní cyklus metadatové migrace ilustuje obrázek v příloze (P 3).

5.5 Migrace dat

Pro manipulaci s daty na cílovém prostředí se používá webová služba Salesforce Bulk API 2.0, jež byla popsána dříve v textu. Pro její obsluhu vzniklo rozhraní *ISfBulkDa-*

taService a realizace rozhraní *SfBulkDataService*. Pro konstrukci HTTP požadavku a konzumaci dat z odpovědi postačila implementace jediné třídy (*SfBatchJob*), jejíž instance v rámci etap datové migrace nabývá různých stavů. Jednoduchost struktury je zapříčiněna použitím protokolu REST u webové služby systému Salesforce. UML diagram zapojených tříd znázorňuje obrázek (Obr. 5.5). Pro zpřístupnění metod datové migrace v třídách vrstvy Controller bylo opět využito návrhového vzoru Dependency Injection.

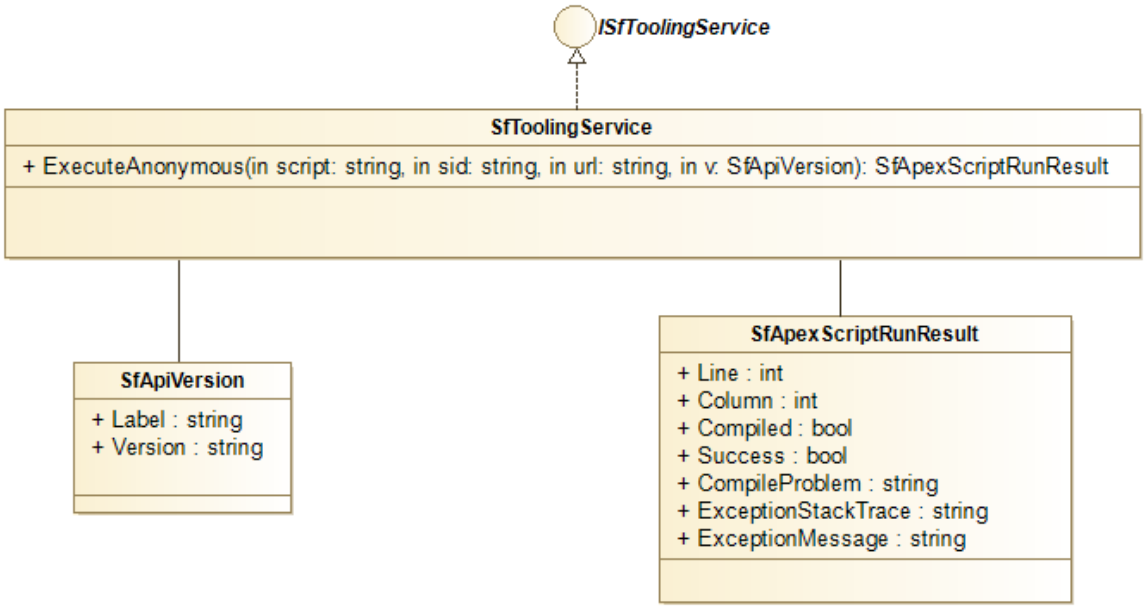


Obr. 5.4 Struktura tříd pro spouštění datových migrací

Datová migrace začíná voláním metody *CreateJobAsync*, která cílové prostředí informuje o tom, že nastane manipulace s daty. Instance třídy *SfBatchJob*, jež je jedním z parametrů, nese informaci o typu operace. Dále se předává přístupový token a URL adresa webové služby. Systém Salesforce vytvoří schránku, jež je naplněna záznamy ze souboru dat pomocí metody *UploadDataAsync*. Ke spuštění procesu zpracování nahraných dat dojde po zavolání metody *RunJobAsync*. Poté je nutné v pravidelných časových intervalech kontrolovat stav manipulace s daty pomocí metody *CheckJobStatusAsync*, která opět vrací instanci třídy *SfBatchJob*. Stav procesu indikuje vlastnost *State*. Došlo-li při zpracování dat k chybě, příčinu lze vyčíst z vlastnosti *ErrorMessage*. Z hlediska uživatelského prostředí se datová migrace jeví stejně jako migrace metadatová. Uživatelovi je po spuštění zobrazen ukazatel průběhu a po dokončení procesu je informován o jeho výsledku.

5.6 Spouštění Apex skriptů

K spouštění uživatelových Apex skriptů se využívá funkcí webové služby Salesforce Tooling API. Služba funguje na principech REST protokolu, což stejně jak v případě datové migrace umožnilo implementaci jednoduchých struktur pro komunikaci s aplikačním rozhraním. Samotný skript reprezentuje pouze datový typ řetězec, který je společně s přístupovým tokenem a adresou webové služby předáván instanci rozhraní *ISfToolingService*. Implementaci jediné nabízené metody *ExecuteAnonymous* realizuje třída *SfToolingService*. Výsledek spuštění skriptu lze získat z vlastností třídy *SfApexScriptRunResult*. Pro zpřístupnění metody třídám vrstvy *Controller* je opět použit návrhový vzor Dependency Injection. Souvislost tříd a rozhraní lze vyčíst z UML diagramu na obrázku (Obr. 5.5).



Obr. 5.5 Struktura tříd pro spouštění Apex skriptů

Narozdíl od datové či metadatové migrace funguje proces provedení skriptu na synchronním principu. HTTP požadavku, který spouští skript na cílovém prostředí, se v odpovědi rovnou vrací výsledek provedení. Není zde nutné kontrolovat výsledek procesu v pravidelných časových intervalech. Výsledek provedení determinuje atribut *Success*. Dojde-li při kompilaci zdrojového kódu k chybě, lze její charakter vyčíst z atributu *CompileProblem*. Pokud se chyba projeví až při vykonávání kódu, systém Salesforce signalizuje chybu, jejíž detaily obsahují atributy *ExceptionMessage* (popis chyby) a *ExceptionStackTrace* (cesta reprodukce chyby). Pro oba chybové stavy platí, že pokud je možné určit chybový řádek a sloupec v rámci skriptu, odkazují na ně atributy *Line* a *Column*.

5.7 Zabezpečení aplikace

V konečné fázi implementace byly provedeny testy zabezpečení systému. První činnost spočívala ve vlastním ověření odolnosti vůči hrozbám vyjmenovaným v teoretické části textu. Osoba testující aplikaci se zde vžila do role útočníka a pokoušela se narušit stabilitu aplikace pomocí praktik jednotlivých hrozeb. Poté byla webová aplikace podrobena automatizovanému bezpečnostnímu testu pomocí nástroje OWASP ZAP ve verzi 2.7.0. Následující text popisuje výsledky bezpečnostních testů a případná opatření.

5.7.1 Autentizace uživatelů

Jak již bylo avizováno, k ověření identity uživatelů ve výsledném systému se využívá technologie ASP.NET Core Identity. Pro zavedení entity uživatele vznikla třída *ApplicationUser*, která byla představena v rámci popisu doménového modelu. Třída se pomocí Entity Framework mapuje do databázové tabulky *AspNetUsers*. Přístupové heslo uživatele je uloženo v zakódované podobě jako otisk původního hesla. Hašování, ověření správnosti a udělení přístupu tak zcela závisí na funkcích technologie ASP.NET Core Identity.

Odpovědnost vývojáře zde spočívá v správném začlenění autentizace do webové aplikace. V případě výsledného systému je ověření identity požadováno pro každou akci kromě přihlášení, jelikož neexistuje jiný případ užití, který by dovoloval anonymitu uživatele. Zmíněného nastavení bylo dosáhnuto pomocí kódu uvedeném na obrázku (Obr. 5.6), který obsahuje třída *Startup* nastavující parametry aplikace během jejího spouštění. Přibude-li v budoucnu případ užití dovolující anonymní přístup, je nutné danou akci označit atributem *AllowAnonymous*.

```
            // Add application services.
            services.AddTransient<IEmailSender, EmailSender>();

            services.AddMvc(
                config => {
                    var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
                    config.Filters.Add(new AuthorizeFilter(policy));
                }
            );
        }
```

Obr. 5.6 Kód vynucující autentizaci uživatele

5.7.2 Autorizace v rámci systému

Ve výsledném systému je implementovaná autorizace za účelem chránění přístupu k datům. Ověřenému uživateli se sice v rámci navigace uživatelským prostředím nabízí pouze jeho vlastní data, avšak je nutné předpokládat, že by se uživatel mohl použitím

nekalých praktik dostat k cizím záznamům. Bez zavedené autorizace by se např. mohlo stát, že útočník uhodne hodnotu identifikátoru záznamu jiného uživatele a pomocí jednoduché modifikace URL adresy k němu přistoupí. První ochranou proti takovým zásahům bylo zavedení identifikátorů s datovým typem řetězec, které jsou automaticky generovány technologií Entity Framework, a to s určitou mírou náhodnosti. Hlavní defenzivní krok představuje implementace autorizace na základě oprávnění. Vznikla třída *AuditableEntityAuthorizationHandler*, která vyhodnocuje udělení přístupu k záznamu na základě předané instance třídy *AuditableEntity* a typu operace (čtení či zápis). Ověření přístupu probíhá na principu porovnání identifikátoru aktuálně přihlášeného uživatele a vlastnosti *CreatedById* u instance z parametru. Stejně jak v případě autentizace je autorizační pravidlo zavedeno do aplikace při jejím spuštění v rámci zdrojového kódu třídy *Startup*. Samotné ověření probíhá v třídách vrstvy *Controller* při práci s daty. Situaci ověření zápisu do objektu typu SfMetadataManifest znázorňuje obrázek (Obr. 5.7). Při neúspěšné autorizaci je zde signalizována vyjímka.

```
AuthorizationResult authResult = await authorizationService.AuthorizeAsync(User,
                                                                            manifest,
                                                                            AuditableEntityOperations.Write);
if (!authResult.Succeeded)
{
    throw new InvalidOperationException();
}
```

Obr. 5.7 Příklad kódu pro kontrolu oprávnění přístupu k záznamu

5.7.3 Aplikovaná opatření proti útoku CSRF

Jak již bylo avizováno v teoretické části práce, uživatel se stává obětí útoku Cross-site Request Forgery, když v systému neúmyslně spustí akci pod svou identitou, a to nejčastěji pomocí odkazu, který mu předá útočník. Technologie ASP.NET Core nabízí ochranu proti CSRF pomocí osvědčené metody zavedení CSRF tokenů.

Token je v systému předáván do každého HTTP požadavku odesílaným metodou POST, který zpravidla spouští modifikaci dat. Je li požadavek sestaven a odeslán pomocí klasického HTML formuláře, token se přikládá automaticky. Obrázek (Obr. 5.8) znázorňuje zdrojový kód formuláře pro vytvoření přístupu k prostředí systému Salesforce, kde lze vidět pole s názvem *__RequestVerificationToken* a hodnotu tokenu.

V systému se však odesílají požadavky také na asynchronní bázi pomocí technologie AJAX. Příkladem je spouštění migrací. V takovém případě se přítomnost CSRF tokenu v HTML dokumentu vynucuje pomocí makra *@Html.AntiForgeryToken()* technologie Razor. Hodnota tokenu je pak získána pomocí skriptu v jazyce JavaScript a vložena do asynchronního HTTP požadavku.

```
▼<form action="/SfOrgCredentials/New" method="post">
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
  ▶<div class="form-group">…</div>
    <input name="__RequestVerificationToken" type="hidden" value=
    "CfDJ8FFF4bJvK0xAq343cUWXpmqCjaZupSR9ioWZ8eNr290_K0KvH0LTL6xDDYsH8MaYkSMvDuozMNeZT
    T2gozjkRg9wD4GEdRPCmr2MoEZzZr48LtBWlTJXCcGxkMU9LPTJXfbhRTOSaVmWXUbvTMAwEsLhESpaPE3
    yo82Hi8-mWcUVWQsYh7XIQM-6W1TeOPAoLQ">
  </form>
```

Obr. 5.8 Přítomnost CSRF tokenu v rámci formuláře

Na straně serveru se zajišťuje kontrola CSRF tokenu pomocí atributu *ValidateAntiForgeryToken* aplikovaným na metodu třídy z vrstvy *Controller*. Při neshodě hodnot server odpovídá chybovým kódem v odpovědi na HTTP požadavek. Příklad aplikace atributu na akci, která vytváří přístupový údaj do systému Salesforce, znázorňuje obrázek (Obr. 5.9).

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> New(SfOrgCredentials orgCredentials)
```

Obr. 5.9 Atribut vynucující kontrolu CSRF tokenu

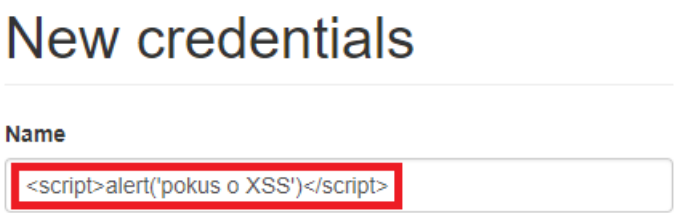
5.7.4 Aplikovaná opatření proti útoku XSS

Útok Cross-site Scripting spočívá ve vložení škodlivého kódu do neošetřeného vstupu webové aplikace. V rámci implementace výsledného systému bylo dbáno na to, aby řetězce vstupující do systému obsahovaly pouze nezbytné znaky. Validace probíhá na straně serveru, kde je vstup porovnáván vůči předepsanému vzoru, který zpravidla představuje regulární výraz.

Povoluje-li vstup aplikace znaky, které by mohly vést ke konstrukci škodlivého kódu, situaci zachraňuje preprocesor Razor, který veškerý výstup v rámci výsledného HTML dokumentu zakóduje. Situaci ilustruje obrázek (Obr. 5.10), kde vstupní pole pro jméno obsahuje pokus o škodlivý kód, a obrázek (Obr. 5.11), kde je hrozba odvrácena pomocí nahrazení závorek a uvozovek jejich zakódovanými podobami. Běžný uživatel nepozná rozdíl, protože prohlížeč znaky interpretuje v čitelné formě.

5.7.5 Aplikovaná opatření proti útoku SQL Injection

SQL Injection je další typ hrozby zapříčiněný neošetřenými vstupy do aplikace. Útok zde může vést k začlenění škodlivého kódu do SQL dotazu. V teoretické části se však nachází informace o tom, že většina moderních technologií obsahuje implicitní ochranu



Obr. 5.10 Potenciálně škodlivý kód útoku XSS

```
<tbody>
        <tr>
            <td>&lt;script&gt;alert(&#x27;pokus o XSS&#x27;)&lt;/script&gt;</td>
            <td>vdvs@vdsvs.cz</td>
```

Obr. 5.11 Zakódovaný výstup technologií Razor

proti SQL Injection. ASP.NET Core v kombinaci s Entity Framework není vyjímkou. Pro komunikaci s databází se v rámci kódu výsledného systému nepoužívá jediný vlastně vytvořený SQL dotaz. Využívá se aplikačního rozhraní technologie Entity Framework, které se stará o bezpečné předávání informací. Příkladem je obrázek (Obr. 5.12), který obsahuje kód pro získání metadatového manifestu z databáze na základě předaného identifikátoru.

```
SfMetadataManifest manifest =
        await dbContext.SfMetadataManifests.Include(m => m.CreatedBy)
                                           .SingleOrDefaultAsync(m => m.Id == manifestId);
```

Obr. 5.12 Využití Entity Framework pro získání záznamu z databáze

5.7.6 Aplikovaná opatření proti útoku Session Hijacking

Při útoku Session Hijacking dochází ke krádeži identity uživatele v systému. Jak již bylo avizováno v teoretické části práce, ochrana proti této hrozbě spočívá v předávání identifikátoru relace mezi klientem a serverem (*session ID*) jiným způsobem než pomocí parametru adresy URL. V případě výsledného systému je hodnota *session ID* uložena v záznamu souboru *cookie*, který nese název *.AspNetCore.Identity.Application*. Záznam nese příznak *HttpOnly*, což značí, že hodnota identifikátoru relace nemůže být získána pomocí skriptu na straně klienta. Snímek zmíněného souboru *cookie* znázorňuje obrázek (Obr. 5.13).

5.7.7 Automatizované bezpečnostní testy

Mimo vlastních manuálních bezpečnostních testů založených na zkušenostech autora, byl výsledný systém podroben ověření bezpečnosti nástrojem OWASP ZAP ve verzi

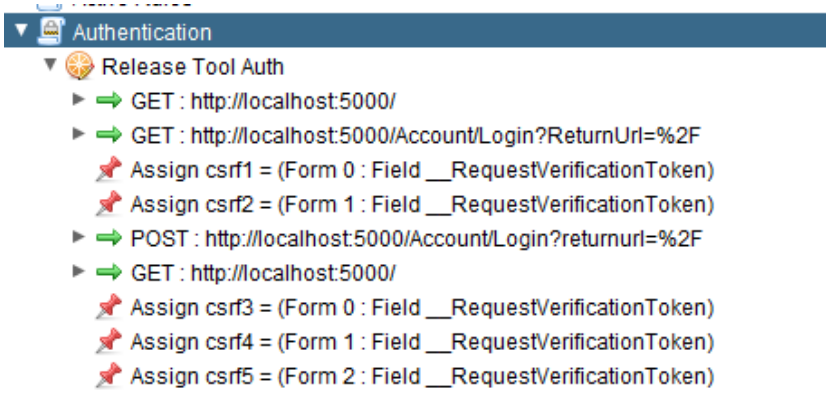
Name	Value	Do...	Pa...	Expires...	Size	HTTP
.AspNetCore.Antiforgery.rAZizMI...	CfDJ8FFF4bJvK0xAq343c...	local...	/	1969-1...	190	✓
.AspNetCore.Identity.Application	CfDJ8FFF4bJvK0xAq343c...	local...	/	1969-1...	635	✓

Obr. 5.13 Záznam souboru cookie obsahující session ID

2.7.0. Jedná se o bezplatný program, který nabízí mnoho možností, jak automaticky procházet webovou aplikaci a testovat její odolnost vůči různým typům hrozeb[1)].

Spuštění automatizovaného testu předcházelo nastavení jeho parametrů. První krok spočíval v nastavení výčtu zkoumaných hrozeb. Při první iteraci testů byla ověřována odolnost vůči všem nabízeným typům, které zahrnovali i výše popsané hrozby.

Dále bylo nutné nakonfigurovat autentizaci vůči systému. V opačném případě by nástroj testoval bezpečnost pouze na stránce pro přihlášení, protože systém nenabízí jiný případ užití, kde by byl povolen anonymní přístup. Pro automatizované testy vznikl dedikovaný uživatel, jehož přístupové údaje byly programu OWASP ZAP předány pomocí makra typu *Authentication*. Jedná se o sezam kroků, které nástroji specifikují, na které URL adrese je možné se přihlásit a jak vypadá HTTP požadavek s přístupovými údaji. Makro znázorňuje obrázek (Obr. 5.14), kde lze vidět sérii HTTP požadavků. První požadavek vede na hlavní stránku webové aplikace, odkud je neověřený uživatel odkázán na stránku pro přihlášení. Následuje požadavek odeslaný metodou POST, který obsahuje přístupové údaje. Poslední záznam značí přesměrování na hlavní stránku webové aplikace po úspěšné autentizaci.



Obr. 5.14 Makro pro zajištění autentizace v nástroji OWASP ZAP

Po nastavení ověření identity testovacího uživatele již bylo možné spustit automatizovaný test. V této fázi by nástroj ověřoval na stránkách, jejichž adresy získá rekurzivním zkoumáním všech odkazů v HTML dokumentech. Výsledná aplikace však nabízí některé funkce, které pracují na asynchronní bázi za použití technologie AJAX.

[1)] Více na `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`

Pro kompletní test bylo nutné nechat aplikaci probádat nástrojem *AJAX Spider*, který v rámci programu OWASP ZAP spustí webový prohlížeč a simuluje uživatelské interakce na různé typy elementů včetně tlačítek, které nejsou v HTML kódu reprezentovány odkazem. Výsledkem je kompletní seznam URL adres pro bezpečnostní test.

Poslední manuální zásah do konfigurace spočíval ve vyčlenění URL adresy, která ve výsledné aplikaci způsobí odhlášení uživatele. V opačném případě by během automatizovaného testu došlo ke ztrátě důkazu autentizace a dále se by se ověřovala bezpečnost pouze na stránce pro přihlášení. Pro výslednou aplikaci stačilo vyčlenit relativní adresu */Account/Logout*.

Výsledky automatizovaného bezpečnostního testu shrnuje tabulka (Tab. 5.2). Obsahuje seznam nalezených hrozeb včetně míry důležitosti. Všechny nálezy referují typy útoků, které nebyly brány v úvahu při analýze systému a vlastním manuálním testování. Následující text popisuje, jak bylo proti nahlášeným bezpečnostním hrozbám zakročeno.

Tab. 5.2 Výsledek testu zabezpečení z programu OWASP ZAP 2.7.0.

	Název chyby	**Stupeň hrozby**
1	Chybějící HTTP hlavička *X-Frame-Options*	Střední
2	Chybějící příznak *HttpOnly* u některých souborů cookies	Nízký
3	Není povolena ochrana prohlížeče proti XSS	Nízký
4	Chybějící HTTP hlavička *X-Content-Type-Options*	Nízký

První nález referuje chybějící ochranu proti útoku Clickjacking. Pokud se v odpovědi na HTTP požadavek nenachází hlavička s názvem *X-Frame-Options* a validní hodnotou, může být webová aplikace vložena do jiné stránky v rámci elementu *frame*, *iframe* či *object*. Dochází tak k oklamání uživatele, který aplikaci používá standardním způsobem a přitom na pozadí neúmyslně spouští akce útočníka. Ve výsledném systému byla hrozba Clickjacking potlačena nastavením hlavičky *X-Frame-Options* na hodnotu *sameorigin*, což umožňuje vkládání stránek webové aplikace pouze v ní samotné.

Druhý nález, jenž se týká chybějícího příznaku *HttpOnly* u některých záznamů v souboru cookie, nevyžadoval aplikace dalších bezpečnostních opatření. Po bližším zkoumání bylo zjištěno, že se jedná o záznamy, které si vytváří technologie ASP.NET Core a které slouží pouze jako mezipaměť pro některé volby uživatele.

Třetí nález doporučuje využít implicitní ochranu webových prohlížečů proti útoku Cross-Site Scripting, kterého lze dosáhnout přítomností HTTP hlavičky *X-Xss-Protection*. Ve výsledném systému byla hlavičce posléze nastavena hodnota „1; mode=block“, což vede k zapnutí ochrany prohlížečem a případnému zastavení vykreslení stránky při detekci hrozby.

Čtvrtý nález referuje chybějící HTTP hlavičku s názvem *X-Content-Type-Options*, která chrání před útokem MIME-sniffing. Bez prevence by útočník mohl zneužít funkce automatické detekce formátu dat u webových prohlížečů a nahrát do systému škodlivý kód, který by maskoval např. jako obrázek. Ve výsledném systému bylo zavedeno opatření, jenž nastavuje hlavičku *X-Content-Type-Options* na hodnotou *nosniff*, což potlačuje automatickou detekci formátu dat a hrozbu MIME-sniffing.

Všechny potenciálně nebezpečné hrozby nalezené pomocí nástroje OWASP ZAP byly potlačeny přidáním HTTP hlaviček do každé odpovědi na HTTP požadavek. Hlavičky jsou implementovány v rámci webové aplikace, což znázorňuje část zdrojového kódu třídy *Startup* na obrázku (Obr. 5.15). Stejného efektu by šlo dosáhnout konfigurací webového serveru, avšak zavedení hlaviček v rámci kódu zajišťuje snadnější přenositelnost výsledné aplikace mezi hostujícími prostředími.

```
app.Use(async (context, next) =>
{
    context.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
    context.Response.Headers.Add("X-Xss-Protection", "1; mode=block");
    context.Response.Headers.Add("X-Content-Type-Options", "nosniff");
    await next();
});
```

Obr. 5.15 Zavedení bezpečnostních HTTP hlaviček

6 Diskuze

Výsledný systém splňuje veškeré předem stanovené funkční i mimofunkční požadavky. Je reprezentován webovou aplikací, která umožňuje spravovat migrační plány obsahující sérii různých migračních kroků. Mezi prostředími systému Salesforce je možné přenášet metadata, modifikovat data a spouštět Apex skripty. V případě existence závislosti automatizované migrace na manuálním zásahu do systému lze v rámci plánu vytvořit a spravovat manuální migrační krok. Pro usnadnění nasazování změn systém nabízí nástroj pro uložení přístupu do prostředí systému Salesforce a nástroj pro tvorbu tzv. metadatových manifestů, které představují seznam komponent pro přenos. Přístup do systému je podmíněn úspěšnou autentizací uživatele, jehož data chrání zavedená autorizační pravidla.

Ačkoliv byly splněny veškeré požadavky, výsledná webová aplikace je pouze prvním prototypem pro řešení problémů migrací mezi prostředími systému Salesforce. Součástí každého životního cyklu systému je další vývoj založený na zpětné vazbě koncových uživatelů. Již během testování první verze aplikace vyvstaly náměty na její zlepšení.

Pro autentizaci by mohla být zvážena možnost využití externích poskytovatelů identity. Uživatel by pak nebyl nucen si pamatovat přihlašovací údaje do výsledného systému, jelikož by se mohl přihlásit pomocí svého již existujícího účtu u dané autority. Jak již bylo avizuje text v teoretické části práce, technologie ASP.NET Identity je na tuto možnost po technické stránce připravena.

Provozu systému by taktéž prospělo, kdyby se pro některé nyní statické seznamy hodnot implementovala jejich automatická aktualizace. Konkrétně se jedná o seznam dostupných verzí systému Salesforce a seznam typů metadat, které jsou v systému uloženy do konfiguračních souborů. V budoucnu by systém mohl na pravidelné bázi získávat dostupné hodnoty pomocí existujících webových služeb. Nebylo by pak nutné při každé aktualizaci systému Salesforce upravovat konfigurační soubory. Systém by automaticky pracoval s aktuálními hodnotami.

Systém by taktéž v budoucnu mohl čerpat metadatové komponenty z dalších zdrojů, konkrétně z verzovacích systémů. Mnoho vývojářů využívá repozitáře těchto systémů pro správu verzí svých aplikací a zpravidla slouží i jako jediný zdroj finální podoby zdrojových kódů. Implementace této funkcionality je však poměrně komplexní a vyžadovala by důkladnou specifikaci požadavků a případů užití.

Jistě existují i další typy systémů, u kterých by šlo přemýšlet o integraci na systém této práce. Vždy záleží na prostředí, kam by se výsledný systém nasazoval, a na zavedených procesech. Zdrojový kód výsledné webové aplikace byl konstruován tak, aby případná rozšíření měla minimální dopad na funkčnost jeho jádra. Systému tak není

bráněno v dalším vývoji, ať už jsou budoucí požadavky jakékoliv.

ZÁVĚR

Tato práce se zabývá analýzou a implementací systému pro správu a vydávání změn do prostředí systému Salesforce. Na základě předem stanoveného zadání byla nejprve provedena analýza současných řešení. Práce představuje hned několik nástrojů pro metadatové i datové migrace. Byly zkoumány jejich nabízené funkce společně s vyhodnocením výhod a nevýhod. Výsledek analýzy konstatuje, že dosud neexistuje systém, který by umožnil všechny typy migrací mezi prostředími systému Salesforce, a zároveň sloužil jako plánovací nástroj pro vydávání změn.

Bylo tedy rozhodnuto pro tvorbu vlastního systému. Z důvodu snadné distribuce byla zvolena reprezentace systému formou webové aplikace. Teoretická část textu popisuje zvažované technologie pro implementaci. Bylo nutné vybrat technologii na straně serveru, která bude odpovídat příchozí HTTP požadavky, vhodný databázový systém a technologie pro prezentaci aplikace na straně klienta. Při výběru byly brány v úvahu zejména možnosti dané technologie, aby umožnily implementaci požadavků na výsledný systém. Dále hrály roli zkušenosti autora práce a také cenová dostupnost technologie. Při výběru byly preferováni kandidáti distribuovány bezplatně jako *open-source*. Výsledný výběr zahrnuje technologii ASP.NET Core 2 na straně serveru, jelikož splňuje všechny stanovené kritéria, a databázový systém PostgreSQL ve verzi 10 pro svou širokou škálu funkcí, které jsou typické pro jinak komerční řešení. Teoretická část taktéž popisuje výhody mapování doménového modelu na tabulky v databázi pomocí technologie Entity Framework a aplikační rozhraní ASP.NET Identity, jež u webových aplikací usnadňuje implementaci autentizace.

Implementaci systému předcházela jeho důkladná analýza. Nejprve byly identifikovány funkční požadavky, které představují detailní popis jednotlivých funkcí systému, a mimofunkční požadavky, jež stanovují omezení některých funkcí a nároky na zabezpečení. Požadavky byly dále detailně rozebrány formou specifikace případů užití. Celkem vzniklo 29 případů užití rozdělených do čtyř funkčních celků. Pro každý případ užití byl sepsán scénář, který detailně popisuje interakci mezi uživatelem a systémem. Na jejich základě byly posléze identifikovány entity figurující v systému. Entity disponují určitými vlastnostmi a vytváří mezi sebou různé typy vztahů, což znázorňuje UML diagram tříd doménového modelu.

Výsledky analýzy sloužily jako podklad pro implementaci systému. Nejprve byly implementovány třídy doménového modelu, které se mapují do tabulek databáze pomocí technologie Entity Framework. Praktická část práce dále popisuje, jak bylo v rámci zdrojového kódu dosáhnuto migrace dat, metadat a spouštění Apex skriptů na prostředích systému Salesforce. Představena je i celková architektura výsledné webové aplikace.

Nedílnou součástí práce je analýza a implementace zabezpečení webové aplikace a bezpečnostních procesů obecně. Teoretická část popisuje známé hrozby a jak se jim bránit. Značná část textu je věnována ověřování identity uživatele a problematice udělování přístupů. Praktická část obsahuje informace o zavedených bezpečnostních opatřeních, včetně způsobu nastavení autentizace pomocí technologie ASP.NET Core a aplikovaná autorizační pravidla. Bezpečnost výsledné webové aplikace byla testována manuálně na základě zkušeností autora, ale také automatizovanou cestou pomocí programu OWASP ZAP ve verzi 2.7.0. Text obsahuje výsledek testu z nástroje a bezpečnostní opatření, které byly posléze zavedeny do zdrojového kódu systému.

Práce splňuje veškeré předem stanovené cíle a zásady pro vypracování. Výsledkem je systém, který nabízí funkce pro správu migračních plánů mezi prostředími Salesforce s možností spuštění všech typů migrací přímo v rámci webové aplikace. Systém navíc disponuje nástrojem pro efektivní tvorbu seznamu komponent, jenž je potřeba při přesunu metadat. Součástí práce je i kritická diskuze, která přináší možnosti pro další případný rozvoj systému.

SEZNAM POUŽITÉ LITERATURY

[1] DOUGLAS, Jeff, 2010. *Salesforce handbook: a newcomer's guide to buiding applications on salesforce.com and the Force.com platform*. United States: Lulu. ISBN 978-144-6108-536.

[2] FREEMAN, Adam, 2017. *Pro ASP.NET Core MVC 2*. London, UK: Apress. ISBN 9781484231494.

[3] CHAFFER, Jonathan, 2013. *Learning jQuery: better interaction, design, and web development with simple JavaScript techniques*. 4th ed. Birmingham: Pack Publishing. ISBN 978-1-78216-314-5.

[4] LEBLANC, Jonathan a Tim MESSERSCHMIDT, 2016. *Identity and Data Security for Web Development: Best practices*. O'Reilly. ISBN 978-1-491-93694-8.

[5] MUELLER, John, 2016. *Security for web developers: using JavaScript, HTML and CSS*. Sebastopol, CA: O´Reilly. ISBN 978-1-491-92864-6.

[6] MUMBAIKAR, Snehal a Puja PADIYA, 2013. Web Services Based On SOAP and REST Principles. *International Journal of Scientific and Research Publications* [online]. [cit. 2018-04-12]. ISSN 2250-3153. Dostupné z: https://pdfs.semanticscholar.org/1f1e/56119acd79c101e7f43f4d7b19f931a6e315.pdf

[7] OBE, Regina O. a Leo S. HSU, 2015. *PostgreSQL: up and running*. Second edition. Beijing: O'Reilly. ISBN 978-1449373191.

[8] POKORNÝ, Jaroslav a Michal VALENTA, 2013. *Databázové systémy*. Praha: České vysoké učení technické v Praze. ISBN 978-80-01-05212-9.

[9] PŘIBYL, Tomáš, 2009. Základní autentizační metody. *ICT Security* [online]. [cit. 2018-04-15]. Dostupné z: http://www.ictsecurity.cz/component/content/article?id=2721

[10] SOMMERVILLE, Ian, 2013. *Softwarové inženýrství*. Brno: Computer Press. ISBN 978-802-5138-267.

[11] SPURLOCK, Jake., 2013. *Bootstrap*. Beijing: O'Reilly. ISBN 978-1-449-34391-0.

[12] Google Authenticator, 2018. *Google Play* [online]. [cit. 2018-04-15]. Dostupné z: https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2

[13] Cybersecurity Market Report, 2017. *Cybersecurity Ventures* [online]. [cit. 2018-04-14]. Dostupné z: https://cybersecurityventures.com/cybersecurity-market-report/

[14] *Tooling API* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2018-04-12].

[15] *Bulk API 2.0* [online], 2017. Version 41.0, Winter -18. San Francisco [cit. 2018-04-12].

[16] *Salesforce.com Winter -18 Release Notes* [online], 2017. San Francisco [cit. 2018-04-12]. Dostupné z: https://resources.docs.salesforce.com/210/latest/en-us/sfdc/pdf/salesforce_winter18_release_notes.pdf

[17] *Bulk API Developer Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2018-04-12].

[18] SOAP API Call Limits, 2018. *Salesforce Developers* [online]. [cit. 2018-04-12]. Dostupné z: https://developer.salesforce.com/docs/atlas.en-us.salesforce_app_limits_cheatsheet.meta/salesforce_app_limits_cheatsheet/salesforce_ap

[19] *Force.com REST API Developer Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2016-11-24]. Dostupné z: https://resources.docs.salesforce.com/sfdc/pdf/api_rest.pdf

[20] *Metadata API Developer Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2016-11-24]. Dostupné z: https://resources.docs.salesforce.com/sfdc/pdf/api_meta.pdf

[21] *SOAP API Developer Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2018-04-11]. Dostupné z: https://resources.docs.salesforce.com/sfdc/pdf/apex_api.pdf

[22] SQL Injection, 2016. *OWASP* [online]. [cit. 2018-04-17]. Dostupné z: https://www.owasp.org/index.php/SQL_Injection

[23] Entity Framework Database First, 2016. *Microsoft Developer Network* [online]. [cit. 2018-04-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/jj206878(v=vs.113).aspx

[24] Introduction to Entity Framework, 2016. *Microsoft Developer Network* [online]. [cit. 2018-04-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx

[25] Add a controller to an ASP.NET Core MVC app, 2017. *Microsoft Developer Network* [online]. [cit. 2018-04-10]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-controller

[26] ASP.NET MVC Overview, 2012. *Microsoft Developer Network* [online]. [cit. 2018-04-10]. Dostupné z: https://msdn.microsoft.com/library/dd381412(v=vs.108).aspx

[27] TIOBE Index, 2018. *TIOBE* [online]. [cit. 2018-04-09]. Dostupné z: https://www.tiobe.com/tiobe-index//

[28] Usage of server-side programming languages for websites, 2018. *W3Tech - Web Technology Surveys* [online]. [cit. 2018-04-09]. Dostupné z: https://w3techs.com/technologies/overview/programming_language/all

[29] Gearset - Features, 2018. *Gearset* [online]. [cit. 2018-04-08]. Dostupné z: https://gearset.com/features

[30] *Data Loader Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2018-04-08]. Dostupné z: https://resources.docs.salesforce.com/204/latest/en-us/sfdc/pdf/salesforce_data_loader.pdf

[31] The Apache Ant Project, 2018. *The Apache Ant Project* [online]. [cit. 2018-04-08]. Dostupné z: https://ant.apache.org/

[32] *Force.com Migration Tool Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2018-04-29]. Dostupné z: https://resources.docs.salesforce.com/sfdc/pdf/salesforce_migration_guide.pdf

[33] *Development Lifecycle Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2018-04-29]. Dostupné z: https://resources.docs.salesforce.com/sfdc/pdf/salesforce_development_lifecycle.pdf

[34] An Introduction to Environments, 2016. In: *Salesforce Developers* [online]. San Francisco [cit. 2016-11-22]. Dostupné z: https://developer.salesforce.com/page/An_Introduction_to_Environments

[35] Entity Framework Code First to a New Database, 2016. *Microsoft Developer Network* [online]. [cit. 2018-04-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/jj193542(v=vs.113).aspx

[36] DOM Based XSS, 2015. *OWASP* [online]. [cit. 2018-04-17]. Dostupné z: https://www.owasp.org/index.php/DOM_Based_XSS

[37] Cross-site Scripting, 2018. *OWASP* [online]. [cit. 2018-04-17]. Dostupné z: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[38] Prevent Cross-Site Request Forgery attacks in ASP.NET Core, 2018. *Microsoft Docs* [online]. [cit. 2018-04-16]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery

[39] Cross-Site Request Forgery, 2018. *OWASP* [online]. [cit. 2018-04-16]. Dostupné z: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[40] Claims-based authorization in ASP.NET Core, 2016. *Microsoft Docs* [online]. [cit. 2018-04-16]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims

[41] Policy-based authorization in ASP.NET Core, 2017. *Microsoft Docs* [online]. [cit. 2018-04-16]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies

[42] Role-based authorization in ASP.NET Core, 2016. *Microsoft Docs* [online]. [cit. 2018-04-16]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles

[43] Introduction to authorization in ASP.NET Core, 2016. *Microsoft Docs* [online]. [cit. 2018-04-16]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction

[44] *Salesforce Security Guide* [online], 2016. Version 38.0, Winter -17. San Francisco [cit. 2016-11-21]. Dostupné z: https://resources.docs.salesforce.com/204/latest/en-us/sfdc/pdf/salesforce_security_impl_guide.pdf

[45] Introduction to Identity on ASP.NET Core, 2018. *Microsoft Docs* [online]. [cit. 2018-04-15]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

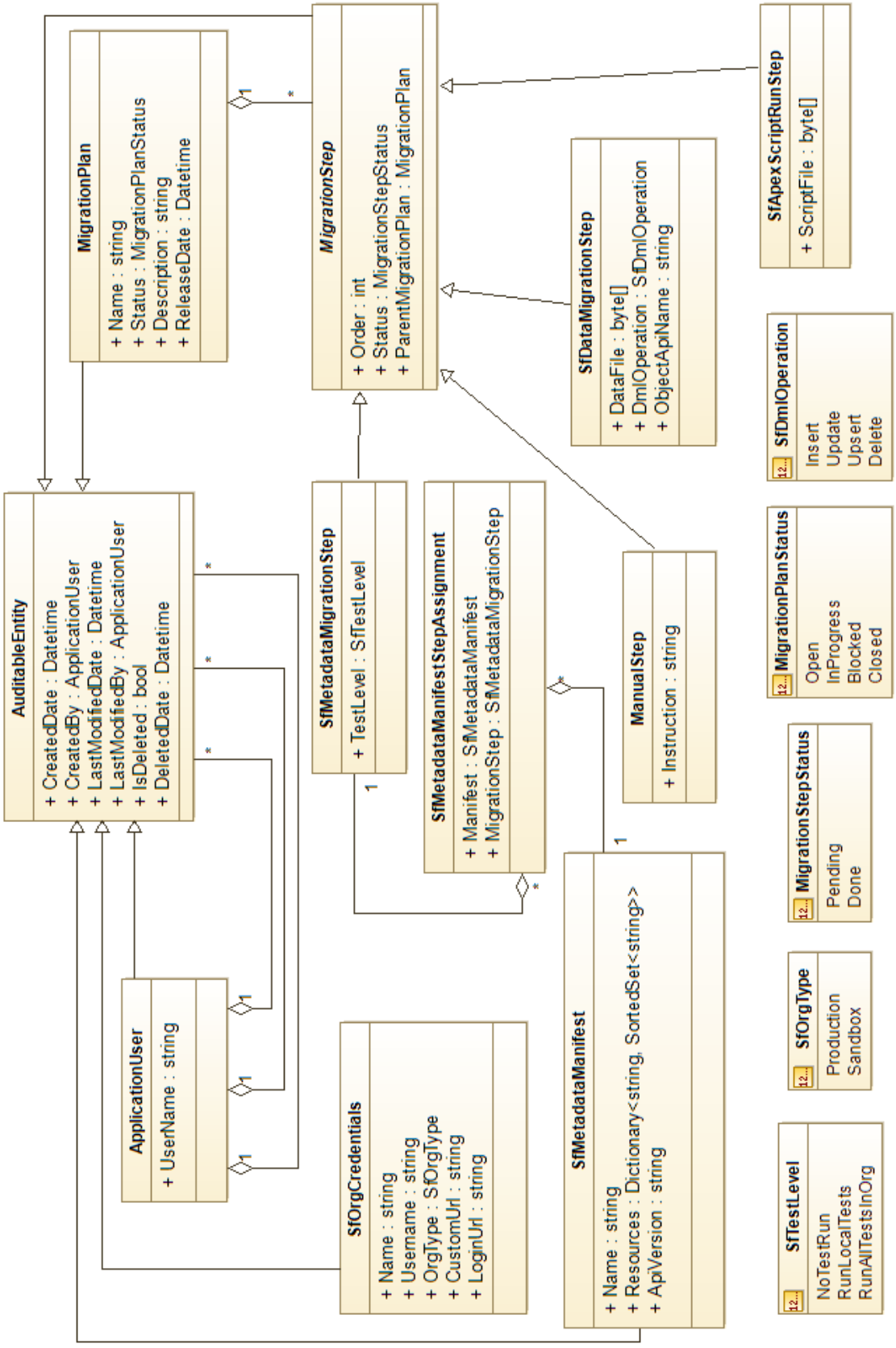
CRM	Customer relationship management
SaaS	Software as a Service
CSV	Comma-separated values
DML	Data Manipulation Language
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
PHP	Hypertext Preprocessor
MVC	Model-View-Controller
JRE	Java Runtime Environment
ASP	Active Server Pages
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
2FA	Two-factor Authentication
CSRF	Cross-Site Request Forgery
XSS	Cross-site Scripting

SEZNAM OBRÁZKŮ

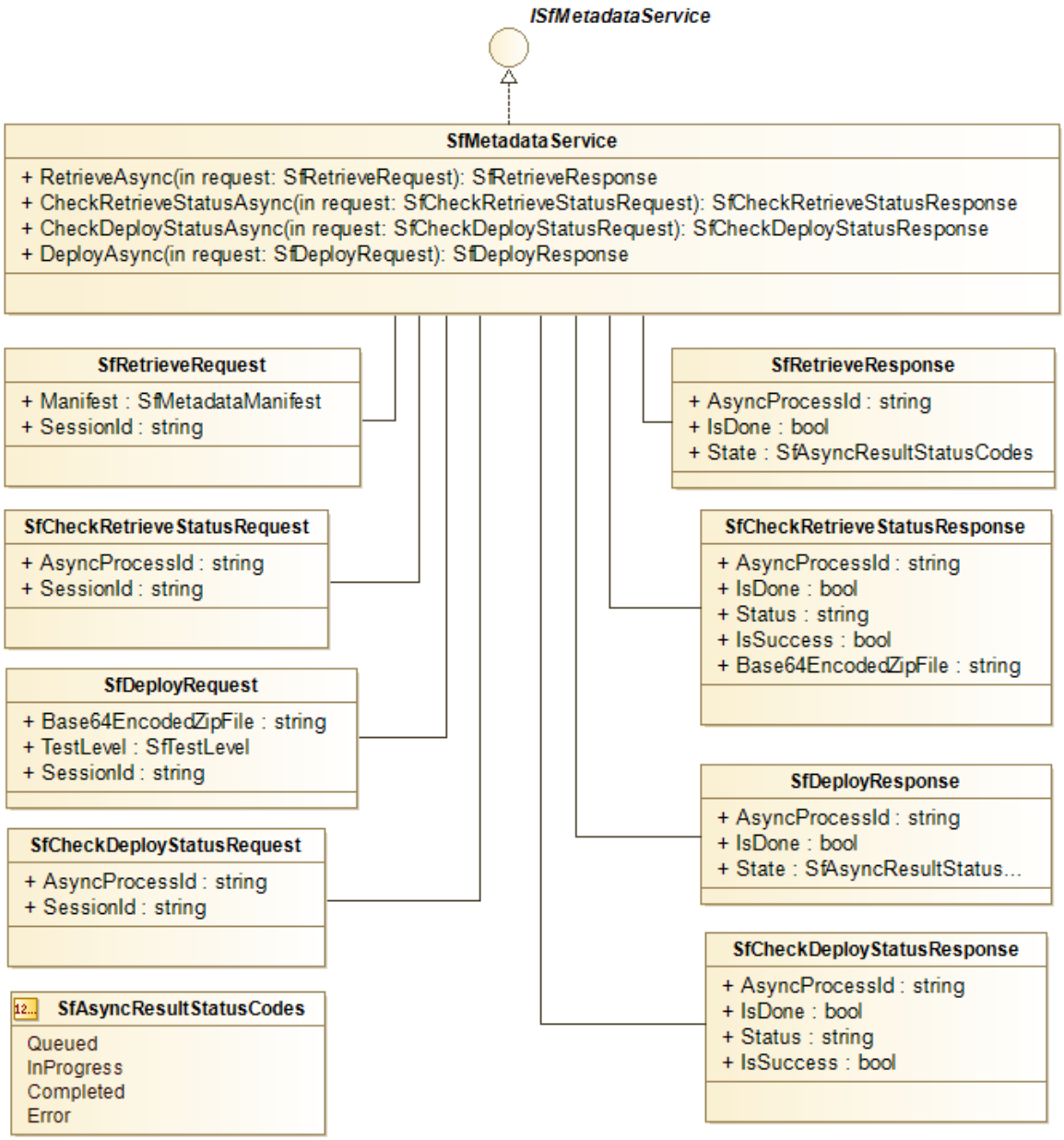
SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I. DOMÉNOVÝ MODEL



Příloha P II. Struktura tříd pro spouštění metadatových migrací



Příloha P III. Životní cyklus metadatové migrace z pohledu uživatelského prostředí