FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Tvorba webové aplikace pro migraci dat
Student:	Jakub Kuchař
Vedoucí:	Ing. Pavel Štěpán
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Webová aplikace pro migraci dat

* DMS migration tool (dále jen DMT) je knihovna založená na technologii .NET Core sloužící k migraci dat mezi spisovými službami. Jedná se o interní nástroj oddělení digitalizace firmy ICZ a.s. Pro konfiguraci knihovny se používá tzv. migrační profil, který definuje zdroj načtení, transformaci, validaci a cíl uložení dat.
* Cílem práce je vytvořit webovou aplikaci, která umožní migraci dat za použití existující knihovny DMT a zároveň zjednoduší konfiguraci tohoto nástroje přidáním grafického rozhraní. Cílovými uživateli aplikace jsou technici, metodici a analytici.
* Analyzujte požadavky na webovou aplikaci a zaimplementujte jejich řešení pro technologii .NET Core. Mezi základní požadavky na aplikaci patří
  - Spuštění migrace pomocí grafického rozhraní
  - Možnost vytvořit či editovat migrační profily
  - Možnost zobrazit si statistiku migrace a logy vygenerované při běhu aplikace
* Výslednou aplikaci vhodným způsobem otestujte a řádně zdokumentujte

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 4. ledna 2021 v Praze.

Bakalářská práce

TVORBA WEBOVÉ APLIKACE PRO MIGRACI DAT

Jakub Kuchař

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: Ing. Pavel Štěpán
22. června 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Pavlovi Štěpánovi za příležitost a cenné rady při tvorbě práce. Dále bych chtěl poděkovat mým kolegům z oddělení digitalizace za vytvoření pracovního prostředí, ve kterém bylo možné bakalářskou práci důkladně vypracovat a zvládnout u toho vykonávat svojí práci zodpovědně. Nakonec bych rád poděkoval mé rodině a nejbližším přátelům, za to, že mi posledních několik let byli při studiích oporou. Bez nich bych se nikdy tak daleko nedostal.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 22. června 2022 .....

Abstrakt

Práce se zabývá implementací webové aplikace pro podporu migrace dat spisových služeb. Výsledná aplikace je založená na technologiích společnosti Microsoft. Pro klientskou aplikaci byla použita technologie Blazor WebAssembly a pro serverovou část ASP.NET Core WEB API obojí ve variantě pro .NET 6. Aplikace k migracím využívá již existující knihovnu firmy ICZ z oddělení digitalizace, která je postavena na technologii .NET Core 2.1.

Oproti existujícímu způsobu migrace, vytvořené řešení umožňuje jednodušší konfiguraci za pomoci konfiguračního editoru. Dále, přidáním grafického rozhraní nad daty z migrační knihovny, poskytuje větší přehled nad procesem migrace.

V příloze lze nalézt všechny analytické dokumenty a instalační příručku. Veškeré zdrojové kódy jsou uložené na přiloženém médiu.

Klíčová slova implementace webové aplikace, elektronická spisová služba, migrace dat spisových služeb, data management systems, data migration tool, C#, .NET 6, ASP.NET Core WEB API, Blazor WebAssemly

Abstract

The thesis deals with the implementation of a web application to support migrations of records management data. The resulting application is based on Microsoft technologies. Client application was build on top of Blazor WebAssembly and sever on ASP.NET Core WEB API, both in .NET 6 variant. Application uses an existing library of ICZ's digitalization department, which is build on .NET Core 2.1 technology.

Compared to the existing way, the created solution enables easier configuration with the help of a configuration editor. Futhermore, by adding graphical interface on top of the data from the migration library, new solution provides a greater control over the migration process.

You can find analytical documents and installation manuals in the appendix. All source codes of resulting application are located on the enclosed medium.

Keywords web application implementation, electronic records management, migration of records management data, data management systems, data migration tool, C#, .NET 6, ASP.NET Core WEB API, Blazor WebAssemly

Seznam pojmů

Seznam zkratek

API	Application Programming Interface
CRUD	Create Read Update Delete
CSV	Comma–separated values
DAL	Data Access Layer
DEPO	Další elektronizace postupů orgánů veřejné moci
DI	Dependency Injection
DMS	Data Management Systems
DMT	DMS Migration Tool
DTO	Data Transfer Object
EF	Entity Framework
ETL	Extraction Transformation Load
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IS	Information System
LTS	Long Term Support
MVČR	Ministerstvo vnitra České republiky
NSESSS	Národní standard pro elektronické systémy spisové služby
ORM	Object Relational Mapping
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SPA	Single–page Application
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
XML	Extensible Markup Language
XSD	XML Schema Definition

Seznam termínů

best practice	osvědčený postup
breaking change	změna mezi verzemi rozhraní software, která znemožní zpětnou kompatibilitu
chromium	svobodný projekt společnosti Google, jehož zdrojové kóde je možné sestavit do prohlížeče
endpoint	koncový bod
Hot Reload	aktualizace zdrojových kódů běžící aplikace, bez ztráty aktuálního stavu běhu
NuGet balíček	balíček obsahující znovu použitelný zdrojový kód distribuovaný pomocí technologie NuGet
reverse engineering	proces odhalení fungování zkoumaného software či informačního systému
singleton	návrhový vzor, definuje jedinečnou instanci třídy přes celou aplikaci
SQL injection	technika napadení databáze vložením škodlivého kódu do databázového dotazu
view	pohled na data databáze definovaný jako databázový dotaz

Kapitola 1
Úvod a cíle bakalářské práce

S příchodem nového zákona DEPO viz MVČR [1], který slouží k podpoře elektronizace státní správy, nově vznikla pro všechny veřejnoprávní původce nutnost vést spisovou službu, která je atestovaná. To znamená, že do roku 2024 musí všichni veřejnoprávní původci přestat používat neatestované spisové služby. Veřejnoprávními původci jsou všichni původci určení zákonem č. 499/2004 Sb., o archivnictví a spisové službě [3, § 3 odst. 1]. Z tohoto důvodu se předpokládá přechod velkého množství zákazníků na jiné systémy. Proto vznikla v sekci DMS firmy ICZ knihovna DMT, která slouží nejen k podpoře migrace neatestovaných spisových služeb, ale i jiných aplikací. Migrační knihovna byla úspěšně použita při několika migracích, jenže byl odhalen problém s velkou náročností konfigurace. Z tohoto důvodu bylo rozhodnuto o vytvoření grafického rozhraní, které bude sloužit ke snadnější konfiguraci a použití knihovny DMT.

Výsledek této práce je určen zaměstnancům firmy ICZ ze sekce DMS, kteří se zabývají a budou zabývat migrací dat spisových služeb na projektech firmy ICZ. Aplikace má především zjednodušit konfiguraci a použitelnost existující migrační knihovny a tím usnadnit migrační proces, který je sám o sobě velmi náročný.

Autor si téma vybral, protože je jedním z cílových uživatelů výsledné aplikace, má zkušenosti s problematikou migrace dat a je jeden z původních tvůrců migrační knihovny DMT. Bakalářská práce se zabývá problematikou migrace a představením aktuálního řešení. Dále se zabývá analýzou, implementací a testováním webové aplikace postavené na nové technologii Blazor WebAssembly od společnosti Microsoft.

Hlavním cílem bakalářské práce je vytvořit webovou aplikaci, která umožní a zjednoduší konfiguraci knihovny pro migraci dat spisových služeb a usnadní uživatelům vyhodnocení průběhu a stavu této migrace. Vypracování tohoto cíle se skládá z menších kroků. Prvním krokem je zhodnocení aktuálního způsobu migrace. To znamená definovat migraci, popsat proces migrace v kontextu této knihovny a její použití při migracích spisových služeb v ICZ viz kapitoly 3 a 4. Dalším krokem je analýza změn, které byly provedeny ze strany migrační knihovny za účelem zjednodušení její konfigurace a integrace s webovou aplikací viz kapitola 5. Třetím krokem je popsání výsledného stavu z úhlu pohledu aplikace samotné. Popis vybrané technologie a architektury aplikace viz kapitola 6. Čtvrtým a nejdůležitějších krokem je samotná implementace aplikace viz kapitola 7. Implementace musí splňovat hlavní cíl celé práce, tím je zjednodušení samostatné migrace. Z tohoto důvodu se bude implementovat editor pro migrační profil, který bude schopný vyhodnotit správnost konfigurace. Dále aplikace musí umožnit uživatelům jednoduché plánování procesu migrace dat a zároveň dát přístup ke statistice zpracování. Posledním a neméně důležitým krokem je otestování aplikace a vypracování její dokumentace viz kapitola 9.

Bakalářská práce a implementace aplikace navazuje na migrační knihovnu DMT firmy ICZ.

Kapitola 2

Spisová služba a zákon

„Pojmem spisová služba označujeme oběh písemnosti v kanceláři, úřadu či instituci, a to od přijetí písemnosti, jejich označení, zapsání, přidělení ke zpracování či vyřízení, podepsání, odeslání a uložení.“ [2, s. 85].

V případě Českých zemí sahá historie spisových služeb až na počátky 18. století, které autorka knihy Archivnictví nazývá „Období kanceláří – období knižních registratur“ [2, s. 86]. Od té doby se způsob provozu spisových služeb několikrát změnil. Až v roce 2004 byla ustálená podoba elektronických spisových služeb, které se provozují do dnes. Tuto podobu definoval zákon č. 499/2004 Sb. o archivnictví a spisové službě.

Spisovou službu je možné vykonávat v listinné, nebo elektronické podobě. Pro většinu veřejnoprávních původců existuje povinnost vykonávat spisovou službu v elektronické podobě. Viz útržek ze zmiňovaného zákona „Veřejnoprávní původci vykonávají spisovou službu v elektronické podobě v elektronických systémech spisové služby“ [3, § 63 odst. 3]. Pouze ve speciálních případech mohou vykonávat veřejnoprávní původci spisovou službu v listinné podobě. Jak bylo zmíněno v úvodu, tak veřejnoprávní původci jsou daní zákonem. Za původce je považován „každý, z jehož činnosti dokument vznikl“ [3, § 2 odst. e].

V nejzákladnější podobě je možné si představit elektronickou spisovou službu jako IS, který umožňuje odborně spravovat dokumenty po celý jejich životní cyklus. Od vzniku dokumentů, tedy přijetí či vytvoření. Až po jejich likvidaci, tedy skartaci nebo archivaci, které se provádějí ve spisovně. Spisovnou je místnost či budova, která je určená k uložení dokumentů a spisů po celou dobu jejich skartační lhůty. Skartační lhůta je období archivace, do doby skartačního řízení. Nakonec skartačním řízením je proces, při kterém se spisy buď archivují, nebo skartují. Ve spisovně je možné dokumenty vyhledávat pro potřeby původce a provádět dané skartační řízení.

Nejdůležitějšími entitami spisových služeb jsou dokumenty a spisy, které jsou definovány, stejně jako další zmíněné entity, v sekci níže pomocí Národního standardu. V případě elektronických spisových služeb je možné vést nejenom elektronické, ale i analogové dokumenty. Ty jsou v případě elektronické spisové služby reprezentovány jednoduchým záznamem, který odpovídá fyzickému dokumentu. Stejně tak je možné vést i elektronické a analogové spisy. Každá ze zmiňovaných entit má unikátní jednoznačný identifikátor napříč systémem a další označení jako je číslo jednací čí spisová značka. Pod těmito identifikátory je možné dané entity v celém systému dohledat.

Elektronické spisové služby bývají typicky propojeny s externími systémy, které nějakým způsobem pracují s entitami spisové služby. Definice tohoto propojení je daná zákonem pomocí Národního standardu viz sekce níže.

2.1 Národní standard

V roce 2009 byl představen nový zákon č. 190/2009 Sb., který upravuje zákon č. 499/2004 Sb. Tento zákon nově přinesl Národní standard pro elektronické systémy spisové služby, zkráceně NSESSS. Daný standard definuje a sjednocuje technické požadavky na výkon spisové služby a podobu rozhraní a dat mezi různými spisovými službami či externími agendami. Externími agendami jsou myšleny aplikace pracující s daty spisové služby přes jejich vystavené API. Zákon byl postupně novelizován podle praktických poznatků provozovatelů spisových služeb až do podoby z roku 2017. Veškeré datové modely je možné nalézt na stránkách MVČR [4].

V kontextu migrace dat spisových služeb jsou nejdůležitějšími výsledky tohoto zákona datové modely, pomocí nichž mohou provozovatelé různých spisových služeb předávat migrovaná data. Tyto modely sice nejsou dokonalé, protože neumožňují při migraci přenést historii entity, ale i tak umožňují přenést většinu potřebných informací.

Pro migraci se používají tyto modely entit (viz definice NSESSS):

Číslo jednací „Číslo jednací je evidenční znak dokumentu v rámci evidence dokumentů, jehož tvar vychází z požadavků jiných právních předpisů a potřeb původce.“ [5, s. 2]

Dokument „Dokumentem je každá písemná, obrazová, zvuková nebo jiná zaznamenaná informace, ať již v podobě analogové nebo digitální, která byla vytvořena původcem nebo byla původci doručena.“ [5, s. 3]

Komponenta „Komponentou v digitální podobě se rozumí jednoznačně vymezený proud bitů tvořící počítačový soubor. V analogové podobě je komponentou dále nedělitelná část dokumentu (průvodní dopis, příloha).“ [5, s. 5]

Spisová značka „Spisová značka je evidenčním znakem spisu (identifikací spisu), pokud tak stanoví jiný právní předpis nebo interní předpis původce.“ [5, s. 8]

Spis „Spis je entita, v níž jsou organizovány dokumenty vztahující se ke stejnému předmětu (věci). Spisy se vyskytují pouze ve věcných skupinách, které neobsahují jiné věcné skupiny nebo typové spisy.“ [5, s. 8]

Typ dokumentu „Typem dokumentu se rozumí věcná charakteristika popisující dokument…Typem dokumentu jsou například *faktury*, *smlouvy* nebo *webové stránky*.“ [5, s. 10]

Zásilka „Zásilka je prostředek pro doručování dokumentů v analogové nebo digitální podobě. Zásilkou je nejčastěji listinná obálka, datová zpráva z informačního systému datových schránek, e-mail, optický disk nebo flash disk.“ [5, s. 11]

Nejedná se o vyčerpávající seznam používaných modelů k migraci.

2.2 Další elektronizace postupů orgánů veřejné moci

V roce 2021 byl schválen nový zákon č. 261/2021 Sb., který mění zákony v souvislosti s další elektronizací postupů orgánů veřejné moci, zkráceně DEPO. Tento zákon přináší spoustu změn, ale v kontextu spisových služeb jsou jedny z nejdůležitějších: nová povinnost atestovat spisové služby a povinnost vést spisové služby elektronicky.

Viz MVČR „Zavedením povinného atestování elektronických systémů spisové služby se vytvoří předpoklad pro kvalitnější výkon spisové služby orgány veřejné moci. Spolu s povinností vykonávat spisovou službu výlučně elektronicky to umožní urychlit zpracovávání podání občanů a firem a zefektivní vnitřní provoz orgánů veřejné moci.“ [1].

S příchodem tohoto zákona se předpokládá, že dojde k velkému odlivu uživatelů z existujících spisových služeb, které se nedokáží přizpůsobit atestaci. Jak bylo zmíněno, tak se jedná o jeden z důvodů vzniku původní migrační knihovny.

Kapitola 3
Migrace

3.1 Definice migrace

V knize Practical Data Migration je migrace definovaná jako „proces selekce, přípravy, extrakce a transformace dat a jejich trvalé přenesení z jednoho softwarového úložiště do jiného“ [6, s. 7]. Z této definice jsou vidět základní kroky migrace.

Selekcí dat je myšlena identifikace zdrojů, jako jsou například databázové systémy, tabulkové soubory, nebo různá API a následná identifikace dat z těchto zdrojů.

Extrakce a transformace dat jsou kroky ve kterých jsou data ze zdroje načtena a převedena do formátu vhodného pro cílový systém. Při těchto krocích se provádí veškeré potřebné úkony k zajištění očekávaného formátu dat pro následný systém. Je–li potřeba, tak se data filtrují, nebo dále nezpracovávají. Je jednodušší a levnější nevhodná data nezpracovat vůbec, než provádět jejich opravu v cílovém systému po nevalidním nahrání. Cílovými úložišti jsou myšleny databázové či informační systémy.

Migrace je složitý proces, jehož náročnost roste s objemem dat, které je potřeba brát při migraci v potaz. U složitějších systémů, kterými jsou například spisové služby, je možné se domnívat, že bude proces náchylný k chybám. V knize Practical Data Migration autor vyjmenovává několik typických problémů [6, s. 9]. Může se jednat o nedostatečnou znalost specialistů na migraci, podcenění velikosti dat, podcenění délky trvání migrace a pohled na migraci, jako na čistě technický problém. Dalšími problémy mohou být nedostatečná analýza migrace, nebo neinformovanost cílového uživatele migrovaného systému. Například se může jednat o situaci, kdy uživatel jasně používá určité funkcionality v jedné spisové službě, které ve druhé chybí. Uživatel si poté může myslet, že migrace nebyla dokonalá a určitá data se ztratila. Tento problém může vzniknout, pokud se před migrací dostatečně nekomunikuje s uživateli, nebo se tento krok úplně přeskočí. Každý ze zmiňovaných problémů je potřeba správně adresovat, aby byla migrace úspěšná.

3.2 Extrakční nástroje

Při migraci se nabízí používat automatizované nástroje. Nejenže přinášejí komfort ve formě automatizovaného spuštění a běhu, ale i kontrolu nad daty, které jsou migrované. V případě migrace spisových služeb se může jednat o miliony záznamů, tudíž je vhodné takové nástroje použít.

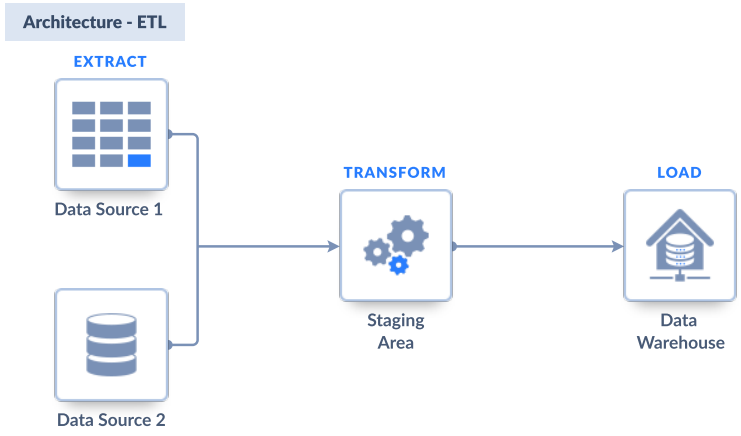
Příkladem jsou nástroje typu ETL, česky nástroje pro extrakci, transformaci a načtení. Z článku na webu Experian [7] je vidět význam této zkratky. Jedná se o názvy kroků, které jsou v případě migrace těmito nástroji vykonávány viz obrázek 3.1.

V prvním kroku se provede extrakce dat z nejrůznějších systémů. Může se jednat o databáze, tabulkové soubory, nebo konkrétní API. V dalším kroku se provede transformace daných dat do

formátu vhodných pro cílový systém. Při transformaci se může aplikovat *business* (obchodní) logika, přečíslování identifikátorů, filtrace nevalidních, nebo nepotřebných informací, doplnění chybějících informací a další. V posledním kroku je provedeno načtení dat do cílového systému. Stejně jako v extrakci se předpokládá široká škála typů cílových systémů.

Tyto nástroje provádějí všechny kroky automaticky, slibují jednoduchou definici a přehled nad migrovanými daty. Z tohoto popisu je jasně vidět jejich využití a výhoda, proto sloužily jako podklad při tvorbě původní migrační knihovny, která slibuje podobné funkcionality viz kapitola 4.

Obrázek 3.1 Architektura ETL nástroje, převzato z `https://skyvia.com/`



Kapitola 4
Aktuální řešení migrací

Migrace, které migrační tým řeší, se většinou týkají spisových služeb, nebo aplikací s nimi spojených. Migračním týmem jsou myšleni všichni aktivní účastníci migrace, kteří jsou blíže rozepsaní v sekci 5.6. Nejčastější variantou je migrace z cizí spisové služby do firemní. Pravidelně se stává, že migrovaná spisová služba nesplňuje Národní standard (viz předchozí kapitola 2). Z tohoto důvodu je potřeba mít robustní řešení, které umožňuje data získat, podle konfigurace transformovat, vyčistit, validovat a poté předat do potřebných systémů. Z tohoto důvodu vznikla migrační knihovna DMT, která všechny tyto funkce umožňuje.

4.1 Migrační knihovna

Aktuálně se k migraci používá migrační knihovna DMT, která je postavená na technologii .NET Core od společnosti Microsoft. Tato knihovna se používá v kombinaci se stejnojmennou konzolovou utilitou, která má za úkol pouze nakonfigurovat a spustit migrační knihovnu.

Ve srovnání s ETL nástroji migrační knihovna přináší podobné funkce. Při migraci je možné definovat zdroj načtení dat, kterým můžou být souborový systém, nejrůznější API, nebo databáze. V případě databáze se používají klasické SQL dotazy, které jsou definovány v nastavení migračního profilu. Při načítání dat z databáze nebo API je potřeba předem definovat identifikátory dat, které se mají k načtení použít. Při načítání se identifikátory vkládají do generovaných dotazů a tím je zaručeno, že se ze zdrojového systému získají očekávaná data. Jedná se o proces selekce dat, ve kterém se generují dotazy s vyplněnými parametry.

Po načtení souborů přichází na řadu předtransformační validace. Ta se vztahuje pouze na data, která byla načtena ze souborového systému v podobě XML souborů, nebo na dotazy ze SOAP API, kde jsou data také reprezentována v XML formátu. Při validaci se používají předem vytvořené XSD šablony, které definují očekávaný tvar dat. Data, která se nepodaří validovat, jsou ze zpracování vyřazena.

Dalším krokem je transformace dat. Transformace je definována jako postupné aplikování různých transformačních kroků k vytvoření jiného či upravení stávajícího migrovaného objektu. Je možné si tuto definici představit jako imperativní programovací jazyk, kde je přesně definována posloupnost příkazů, která vede ke zdárné transformaci. V útržku kódu 4.1 je vidět jak taková transformace vypadá.

Transformace jsou rozdělené podle skupin položek, se kterými pracují. Může se jednat o seznamy, jednotlivé položky, nebo hodnoty těchto položek. Pro každou z těchto skupin jsou definované základní transformační akce: přidání či vytvoření, kopírování, přesunutí, smazání, nahrazení hodnotou ze slovníku, přenesení mezi dvěma komplexními objekty, nebo nahrazení hodnoty pomocí regulárního výrazu. Konkrétně se jedná o dvacet dva různých transformačních akcí.

Výpis kódu 4.1 Přetypování písemnosti v transformaci

```
...
<SetValue field=".TempObject.Dok_Typ" value="{@typVD}"/>
<SetValue field=".TempObject.Dok_SpsClass" value="VlastniPisemnost"/>
<ConditionalGroup>
  <Conditions>
    <IsField field="Data.ProfilDokumentu.Doruceni"/>
  </Conditions>
  <Actions>
    <SetValue field=".TempObject.Dok_Typ" value="{@typDD}"/>
    <SetValue field=".TempObject.Dok_SpsClass" value="DorucenaPisemnost"/>
  </Actions>
</ConditionalGroup>
<CopyValue sourceField=".TempObject.Dok_Typ"
      targetField=".TempObject.Dok.TypDokumentu"/>
...
```

Následujícím krokem je validace po transformaci, která se na rozdíl od validace před transformací aplikuje na všechny migrované objekty. Definice této validace je podobná definici transformace. Hlavním rozdílem je, že se akce validace definují především pro hodnoty položek. Počet akcí validace je pouze osmnáct. Za zmínku také stojí, že transformace a validace po transformaci je agnostická na typ objektu, se kterým pracuje. Nezáleží na tom, jestli se jedná o jednoduchou či komplexní třídu, nebo XML fragment. Přístup k nim by měl být vždycky stejný.

Poté následuje export dat z aplikace do cílového systému. Podporovanými cílovými systémy jsou různá API a souborové systémy. Při transformaci je potřeba data připravit do správného formátu pro cílový export, proto není jednoduché použít stejnou transformaci pro různé cílové systémy. Po exportu dochází k poslední validaci dat, která stejně jako před transformací, probíhá pouze na datech ve formátu XML.

4.2 Migrační profil

Veškeré fáze migrace popsané v sekci 4.1 jsou definovány a popsány pomocí migračního profilu. Migrační profil je reprezentován XML souborem, který má pevně definovanou strukturu podobně jako jiné konfigurační soubory. Každý XML element, který je součástí validního migrační profilu, reprezentuje existující konfigurační třídu migrační knihovny. Při migraci proběhne deserializace migračního profilu na tyto konfigurační objekty, které následně řídí běh migrace.

Součástí migračního profilu jsou i objekty jako je migrační slovník, či migrační proměnná. V obou případech se jedná o pomocné struktury, které nějakým způsobem usnadňují proces migrace. Migrační slovník i proměnná jsou blíže popsány v sekci 5.7, která je součástí implementační analýzy.

4.3 Migrace spisových služeb

Migrace spisových služeb je, stejně jako už definovaná migrace, proces, při kterém se přebírají data ze zdrojové spisové služby, transformují se do podoby vhodné pro cílový systém, validují se a následně se nahrávají do cílové spisové služby.

V případě takové migrace hraje velkou roli, zda zdrojový IS splňuje Národní standard či nikoli. V prvním případě bývají migrace značně jednoduší, protože stačí pouze přebrat data exportovaná ze zdrojové spisové služby v Národní standardu a podle předdefinovaných způsobů migrace data upravit a nahrát do cílového systému. Tato představa je hodně zjednodušená, jak už

bylo zmíněno, je potřeba data ještě validovat, doplnit o chybějící hodnoty, přemapovat číselníkové položky pro cílový systém a správně nastavit cílový systém. Ale z úhlu pohledu procesu migrace se jedná o jednodušší variantu.

V druhém případě se naráží na problém toho, že spisová služba nepodporuje Národní standard. Pokud zdrojový IS umožňuje data exportovat, tak je možné předpokládat, že ve formátu, který migrační tým nezná. Také je možné předpokládat, že tento formát nebude obsahovat všechny potřebné informace, které jsou k migraci potřeba. V takovém případě je bezpečnější, i když mnohonásobně složitější, migrovaná data získat ze zdrojového systému na přímo. To většinou vypadá tak, že se při migrační analýze provede *reverse engineering* zdrojové spisové služby či jejího databázového systému. Při tom se lokalizují všechna místa, kde se data vyskytují, v jakém formátu jsou uložena a jaké chyby obsahují. Potom se definuje mapování z tohoto formátu na formát vhodný k migraci, například NSESSS. Tímto způsobem jsou data získávána především z databáze zdrojového systému.

Současně se řeší příprava cílového systému. Tato příprava vychází ze stejné migrační analýzy, ve které je rozebráno, jakým způsobem uživatelé do teď zdrojovou spisovou službu používali a jaké očekávají spisové plány, typy dokumentů a jiné nastavení spojené se spisovou službou. Tato nastavení jsou současně brána v potaz při přípravě migrace. Například, je–li odstraněný typ dokumentu, který je umístěný na migrovaných datech, tak je potřeba při transformaci takového objektu provést přemapování daného typu na typ očekávaný v cílové spisové službě. To se týká i jiných číselníkových hodnot. Nesmí se zapomenout na to, že migrace není pouze proces přenesení dat z jednoho systému do druhého, ale také příprava těchto systémů na celý proces.

4.4 Příprava a postup migrace

Příprava migrace se dá rozdělit na více fází: první fáze je analýza zdrojového systému, druhá fáze je příprava a testování migrace, třetí fáze je migrace v produkci a poslední fází je oprava.

V prvotní fázi se členové migračního týmu podílejí na tvorbě migrační analýzy, která slouží jako podklad pro nastavení a dostatečné dimenzování cílového systému. Tato migrační analýza slouží také jako podklad pro vytvoření migračního profilu či profilů, které se pak v následující fázi testují. V této fázi společně spolupracují především analytici, metodici a programátoři.

Po schválení analýzy cílovým uživatelem se přechází na další krok, kterým je příprava migrace a testování migrace. V této fázi je techniky a metodiky připraven cílový systém v testovacím prostředí. Programátoři společně s analytiky pracují na migračním profilu a testují migraci dat do testovacího prostředí. V této fázi je kritické najít všechny chyby a odladit migraci.

Po přípravě cílového systému a migrace do cílového systému přichází den D. V této fázi je odstaven zdrojový systém a přechází se do fáze migrace. Pokud je potřeba, tak se data nějakým způsobem před připraví. Příkladem může být přesunutí dat do souborového systému serveru, ze kterého budou data migrována. Případně příprava zdrojové databáze přidáním podpůrných tabulek a *views*. Následně technik po balíčcích provádí migraci a postupně informuje o zpracování. V této fázi se migrační knihovna používá nejvíce. Veškeré kroky migrace jsou zaznamenány v databázi migrační knihovny z důvodu oprav, které se aplikují v následující fázi. Pokud je potřeba, tak fáze migrace pokračuje i po přechodu cílového systému do produkčního zpracování. V takové chvíli se migrace vykonává v předem domluvených migračních „oknech“ do doby, než je celý systém migrován. Migračními okny se rozumí časové bloky, ve kterých je možné provádět migrace, domluvené mezi migračním týmem a cílovým uživatelem.

Poslední fáze migrace nastává, jakmile je dokončena migrace všech bezproblémových dat. V této fázi jsou nalezeny a postupně opraveny všechny problémy. Může se jednat o reklamace ze strany uživatelů cílového systému, chyby nalezené při transformaci, nebo chyby importu dat do cílového systému. Tato fáze je blíže popsaná v sekci 5.1.4.

4.5 Problémy s tímto postupem

Migrace a její příprava jsou komplikované procesy, u kterých je třeba řešit několik problémů:

Problém s konfigurací migrace Konfigurace migračního nástroje je problematická. To je dáno především tím, že veškerá konfigurace aktuálně probíhá přes textový editor, který žádným způsobem nevaliduje migrační profil. Programátor tudíž nemá jistotu, že je profil správně připravený do doby, než spustí migraci. Není problém, že migrační profil zvládne nakonfigurovat pouze programátor, protože služba, kterou firma nabízí je migrace, nikoli migrační aplikace. To neznamená, že by konfigurace měla být náročná. Přidáním editoru migračního profilu s validací profilu a upozorněním na chyby, by se měla příprava migračního profilu zjednodušit.

Problém s konfigurací slovníků Migrační slovníky jsou v tuto chvíli vytvářeny ručně. To je možné pomocí SQL dotazů, které se provádějí nad databází migrační knihovny. Případně jednodušeji pomocí konzolové utility, která importuje CSV soubor s před připraveným slovníkem do databáze migrační knihovny. Po nahrání slovníku do aplikace není jednoduché dělat jakékoli změny. Navíc není možné při vytvoření migračního profilu ověřit, že slovník použitý v transformaci či validaci existuje a obsahuje položky použité při migraci. Oba problémy budou odstraněny přidáním editoru pro migrační slovník s možností importu a exportu z migrační aplikace. Dále přidáním validace existence slovníku a jeho položek do editoru migračního profilu.

Problém s přehledem Aktuálně jsou všechny informace o zpracování ukládány do databáze migrační knihovny. Proto byly bližší informace z migrační knihovny získávány těmito způsoby: ručním exportem dat pomocí SQL dotazů nad databází a přenesení do aplikace Excel přes uživatelskou schránku, prohlížením vybraných tabulek databáze pomocí nástroje DB Browser for SQLite, nebo použitím aplikace LINQPad pro získání dat z databáze a jejich uložení do CSV souboru. Tudíž neexistuje jednotná cesta jak informace získat a pracovat s nimi. Přidáním jednotného přístupu v podobě okna pro výběr dat s možností filtrace obsahu a exportu do používaných formátů bude sjednocený způsob hledání chyb migrace a vyhodnocení zpracování.

Chybějící automatizovaná migrace Všechny migrace bylo do teď nutné spustit ručně a čekat do konce zpracování. To znamená, že v případě migrací, které probíhaly celý den, bylo nutné zajistit technika pro migraci. Práce technika vypadá tak, že se připojí na stanici s migrační aplikací a zahájí migraci. Pokud je potřeba a jsou domluveny migrační okna, tak se technik ráno znovu připojí a migraci dočasně přeruší. Jedná se o proces, který není vždycky snadno automatizovatelný. Příkladem může být migrace do aplikace, která data nahrává sama a je potřeba jí předem připravit balíčky, které sama importuje. V případě migrace systému, kde přenos do aplikace dělá migrační knihovna, se nabízí mít celý tento proces automatizovaný. To znamená bez nutnosti zapojovat technika. To by měla umožňovat automatizovaná migrace.

Všechny tyto problémy vyřeší aplikace na migraci dat, která je blíže popsaná v analýze viz následující kapitola.

Kapitola 5

Analýza plánované aplikace

Webová aplikace, která je popsána v této vývojové analýze, slouží k podpoře migrace v oblasti spisových služeb. Aplikace si dává za cíl zjednodušit migraci dat. Z tohoto důvodu se autor v této části analýzy zaměřuje na reálné procesy a přípravu migrace, které se pokouší zjednodušit přidáním nových funkcí nad aktuální migrační knihovnu. Funkce, které aplikace poskytuje jsou:

- zjednodušení přípravy migračních profilů
- zjednodušení aktu migrace dat – tím je myšleno spuštění a zastavení migrace
- přidání automatické migrace
- zjednodušení vyhodnocení kvality migrace

5.1 Hlavní procesy

Níže jsou popsány hlavní procesy migrační aplikace. Nejedná se o vyčerpávající seznam, ale o procesy, které postihují nejširší použití celé aplikace. Diagramy procesů jsou uvedeny v příloze C.

5.1.1 Příprava migračního profilu a testování

Přípravou migračního profilu jsou myšleny všechny činnosti od počáteční migrační analýzy přes popsání transformace dat až po připravení a otestování migrace v testovacím prostředí. Do doby, než jsou provedeny testy, není možné považovat migraci za funkční.

Při přípravě migrace je zainteresováno několik osob. Pokud se jejich zodpovědnosti rozdělí, tak je možné určit dvě skupiny: analytik a programátor. Analytik je zodpovědný za analýzu zdrojových dat a popsání potřebných úprav při jejich migraci. Programátor je zodpovědný za přípravu nástrojů podporující migraci, především DMT.

Migrační analýzou je myšlena kontrola zdrojových dat, případné reverzní inženýrství zdrojového systému, které odhaluje bližší informace o migrovaných datech, ověření či pochopení datových struktur a namapování těchto struktur na cílové systémy migrace. Dále se jedná o popsání všech změn, které budou na datech při migraci provedeny, aby bylo možné data nahrát do cílového systému. Na takové analýze spolupracují analytik a programátor společně s cílovými uživateli, protože ti většinou vědí, jak se zdrojový systém chová a je možné s nimi domluvit očekávané chování cílového systému. Po dokončení je analýza předána cílovému uživateli na schválení.

5.1.2 Příprava nastavení a produkčního prostředí

Přípravou nastavení a produkčního prostředí jsou myšleny všechny činnosti spojené s přípravou stanice, nebo prostředí, na kterém poběží migrace. Může se jednat o kontrolu a zřízení prostupů pro migrační nástroj, instalace aplikací, podporující migraci a nastavení těchto aplikací. Jedná se o činnost, kterou má na starosti technik. Může se jednat o stejnou osobu, jako je programátor, ale je potřeba rozlišit zodpovědnosti obou rolí.

5.1.3 Časovaná migrace

Časovaná migrace, nebo též automatizovaná migrace, je jeden ze dvou způsobů, jakým se dají v aplikaci migrovat data. Druhým je manuální migrace, která se liší pouze tím, že je potřeba migraci ručně spustit i vypnout. Časovaná migrace umožňuje migraci dat v přesně definovaných časových oknech, která jsou domluvená se stranou zodpovědnou za cílové prostředí migrace.

5.1.4 Detail opravy dat

Oprava dat je prováděna až po dokončení první fáze migrace. V první fázi jsou migrována všechna bezproblémová data do cílového systému. Poté následuje kontrola chybovosti a následně reakce na chybovost. V této fázi spolupracují podle potřeby analytik, programátor i technik.

Před migrací je se zákazníkem domluvena hranice chybovosti, kterou je při migraci nutné dodržet. Každý nemigrovaný objekt znamená pro cílové uživatele manuální činnosti navíc, které mohou být několikanásobně složitější a dražší. Pokud je to možné, tak je potřeba všechny chyby systematicky vyřešit.

Při migraci se v datech typicky objevují následující chyby:

Chyba migrační analýzy Nejméně pravděpodobná, ale potenciálně nejvíce problematická chyba. Jedná se o situaci, kdy dojde k přehlédnutí závažných skutečností, ze kterých se při migraci není jednoduché či možné vrátit.

Chyba kroku migrace Jedná se o chyby, které nastaly při manuální činnosti technika při migraci. Proces migrace je většinou vymyšlen jednoduše a plně popsán, takže by k chybám docházet nemělo.

Chybná zdrojová data Jedná se o chyby, kdy nejsou migrovaná data před migrací ve validním stavu. Tudíž je není možné migrovat, nebo by jejich migrace vedla k chybě. Většinu chyb je možné lokalizovat už při migrační analýze. Proces migrace a migrační aplikace umožňují odhalit chyby ještě před nahráním dat do cílového systému pomocí předdefinovaných validací.

Chyba v cílovém systému Tato chyba se vyskytuje ve dvou variantách. První a častější z nich je špatná konfigurace cílového systému. Například chybějící uživatelé, nebo hodnoty číselníků. Druhou chybou bývá problém v migračním API cílového systému. Tato chyba se špatně odhaluje a vyžaduje kontrolu a zásah osob zodpovědných za cílový systém.

5.2 Funkční požadavky

Ze zmíněných procesů aplikace a ze zkušeností migračního týmu vzešly následující požadavky na aplikaci.

F01 – Správa migračního profilu

Migrační profil je jádro celé migrace. Umožňuje definici všech kroků migrace. Těmito kroky jsou: definice zdroje dat, validace před nahráním, transformace dat, validace dat po transformaci, export dat do cílového systému a validace dat po exportu.

Z komplexnosti migračního profilu vyplývá, že je důležité mít nástroje na jeho správu. Součástí správy migračního profilu jsou tyto funkce: přidání a odebrání migračního profilu z aplikace, svázání a rozvázání spojení migračního profilu s migrační úlohou, import a export migračního profilu a editace migračního profilu pomocí editoru.

F02 – Editor migračního profilu

Jelikož se migrační profil obtížně konfiguruje, je potřeba mít robustní nástroj na jeho údržbu. Tím je editor migračního profilu. Jedná se o jednu z nejdůležitějších součástí aplikace.

Editor umožňuje upravovat profil stejně jako jsou uživatelé zvyklí. To je přes klasický textový editor. Dále přidává několik nových funkcí, které doposud nebyly možné:

- Zvýraznění validní syntaxe profilu: klíčová slova, proměnné, cesty k položkám a další věci podle inspirace z klasických IDE.
- Zvýraznění nevalidní syntaxe s informací o konkrétním problému.
- Našeptávání hodnot k vyplnění podle kontextu: klíčová slova, cesty k položkám transformace, proměnné, nebo slovníkové hodnoty.

Kontrola validity migračního profilu funguje dvoukrokově. V prvním kroku se ověří, že má migrační profil validní a očekávanou XML strukturou. Pokud by tomu tak nebylo, tak je uživatel informován o problémech v profilu a kontrola zde končí. V druhém kroku se kontroluje správnost obsahu dokumentu. Například existence použitých proměnných, nebo existence slovníků a hodnot slovníků použitých při transformacích.

F04 – Správa slovníků migrace

Migrační slovníky slouží pro získání jedné a více hodnot podle předaného klíče. Slovníky se používají v migraci při transformaci a validaci dat. V migrační aplikaci může být najednou více aktivních slovníků. Správou slovníků je myšleno: přidání a odebrání slovníků, ruční editace slovníků a import či export slovníků. Nejčastějšími příklady slovníků jsou překlady uživatelských jmen a typů dokumentů mezi spisovými službami.

F11 – Správa zpracování migrace s možností běhu více úloh

V případě migrace je možné naplánovat více souběžně běžících úloh. Každá úloha může mít více než jeden profil, ale není možné použít jeden profil pro více migračních úloh. Správou zpracování je myšleno:

- přidání a odebrání úlohy
- propojení úlohy s profily
- aktivace a deaktivace úlohy, aby se automaticky nespouštěla
- přidání a odebrání automatického spouštění úlohy s definovanou délkou běhu
- ruční spuštění a zastavení úlohy

F12 – Spuštění migrační úlohy ručně

Předdefinované úlohy je možné ručně spouštět i když nemají aktivní automatické spuštění. Instance migrační úlohy může v jednu chvíli běžet pouze jedna.

F13 – Plánované časované spuštění úlohy migrace

Úloha může mít předdefinované automatické spouštění. Tím je myšleno mít zadefinované datum a čas, ve kterém se úloha aktivuje a bude vykonávat migraci. Za toto spouštění budou zodpovědné *triggers* (spouštěče).

Úloha může mít více přednastavených *triggers*. Každému z nich je možné definovat čas běhu při spouštění úlohy. Časem běhu se myslí jak dlouho po aktivaci daná migrace poběží. Důvodem je potřeba definice migračních oken, ve kterých je migrace aktivní. V případě migrace aktivního systému je možné předcházet přetěžování infrastruktury a cílového systémů, které se při migracích vyskytuje.

F14 – Zastavení běžících úloh migrace

V zobrazení aktuálně běžících migrací je možné aktivní migrace zastavit. Migrace není možné pozastavit, ale pouze zapnout a vypnout. V případě zastavení se provádí takzvané *graceful* vypnutí. To znamená, že se nejdříve dokončí migrace objektu a poté ukončí úloha celé migrace.

F15 – Souhrnný přehled aktuálně běžících migrací

Aplikace umožňuje zobrazit seznam aktuálně běžících migrací. Odtud je možné migraci vypnout, zobrazit bližší informace a statistiku za posledních sedm dní, nebo přejít do nastavení úlohy.

F21 – Zobrazení záznamů ze zpracování migrací

Pro zobrazení záznamů je možné použít speciální vyhledávání definované ve správě logů. Při vyhledávání je možné vyplnit bližší informace jako jsou datum, úroveň, kontext, stav migrovaného objektu nebo správa záznamu. Při vyhledávání záznamů se používá stránkování obsahu pro ušetření zdrojů aplikace a rychlejší zobrazení informací.

F22 – Export chyb z migrace

Po zobrazení záznamů (viz předchozí funkce) je možné tyto záznamy exportovat do souboru, nebo do uživatelské schránky. V případě exportu není použito stránkování, takže jsou exportována veškerá data podle nastaveného filtru.

5.3 Technické požadavky

N01 – Podpora vícevláknového a víceúlohového zpracování

Migrační knihovna DMT podporuje vícevláknové zpracování. To znamená, že při vytváření webové aplikace byl zachován vícevláknový asynchronní běh s přidanou možností mít spuštěných více migračních úloh najednou.

N02 – Zjednodušení vytváření migračního profilu

Přidání editoru a validace pro migrační profil umožňuje větší přehled nad vytvářeným profilem. To vede ke zvýšené efektivitě při jeho vytváření.

N03 – Multiplatformnost

Použitím nejnovější verze .NET 6, bylo dosaženo podpory běhu aplikace na několika platformách. Konkrétně se jedná o operační systémy Windows 7+, Windows Server 2012+, macOS 10.15+ a několik distribucí Linux, kterými jsou například Debian 10+, Fedora 33+ a Ubuntu 16.04+ [8].

N04 – Konfigurace aplikace

Aplikaci je možné konfigurovat na těchto úrovních:

Webové rozhraní Základní nastavení aplikace a připojení k databázi.

Migrační knihovna Základní nastavení migrační knihovny, omezení velikosti nahrávaných dat, použitá databáze a cesty k použitému slovníku pro migraci.

Logovací knihovna Veškeré nastavení pro logovací knihovnu log4net, která je použitá k logování v celé aplikaci.

Ve všech případech je změna nastavení automaticky propsána do příslušných částí aplikace bez nutnosti restartovat.

5.4 Pokrytí požadavků případy užití

V tabulce 5.1 je vidět, jak byly pokryty funkční požadavky a případy užití. V příloze A je možné nalézt výpis a popis všech případů použití.

5.5 Obrazovky

V příloze B je možné nalézt navržené obrazovky pro migrační aplikaci. Každá z obrazovek implementuje konkrétní funkční požadavky. Všechny funkční požadavky jsou pomocí obrazovek pokryty. Jedná se o následující obrazovky a požadavky:

Dashboard Hlavní obrazovka, na které jsou vidět aktuální informace o běhu migrací. Těmi jsou informace o aktuálně běžících úlohách, naplánovaných úlohách a statistika za posledních sedm dní. Obrazovka implementuje požadavek „F15 – Souhrnný přehled aktuálně běžících migrací“.

Seznam migračních profilů Obrazovka, na které je možné nalézt seznam všech existujících profilů v migrační aplikaci. S těmito profily je možné manipulovat podle požadavku „F01 – Správa migračního profilu“.

Editor profilu Obrazovka s editorem pro migrační profily. Implementuje požadavek „F02 – Editor migračního profilu“.

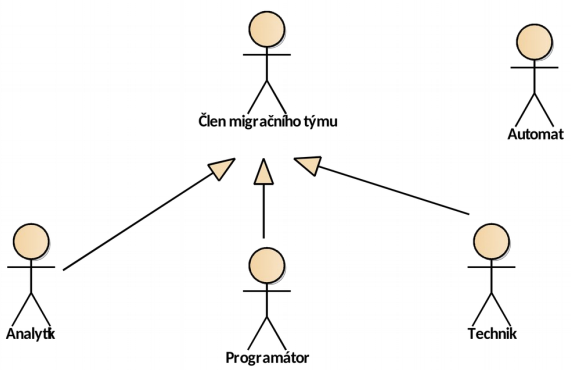
Slovníky Obrazovka, na které je možné vidět a pracovat se všemi migračními slovníky aplikace podle požadavku „F04 – Správa slovníků migrace“.

Seznam úloh Obrazovka, na které je možné najít všechny migrační úlohy a spravovat je podle požadavků: „F11 – Správa zpracování migrace s možností běhu více úloh“, „F12 – Spuštění migrační úlohy ručně“, „F13 – Plánované časované spuštění úlohy migrace“ a „F14 – Zastavení běžících úloh migrace“.

Výpis chyb Obrazovka, na které je možné vyčíst všechny záznamy z migrací podle požadavků: „F21 – Zobrazení záznamů ze zpracování migrací“ a „F22 – Export chyb z migrace“.

5.6 Aktéři

Obrázek 5.1 Aktéři migrace



Analytik Jedná se o osobu zodpovědnou za kontrolu dat, která se mají migrovat a přípravu podkladů pro vytvoření migračního profilu. Tím je myšleno popsání tvaru dat zdrojového systému a popsání změn, které je potřeba v datech udělat. Dále je zodpovědný za přípravu slovníků sloužících k přemapování hodnot z cizího systému. Analytik přímo nepracuje s migračním nástrojem, ale zasahuje do procesu jiných členů migračního týmu.

Programátor Je osoba zodpovědná za přípravu migračního profilu, kterým se řídí DMT při migraci. Programátor má největší vliv na kvalitu přebraných dat, protože je může systematicky upravit a přizpůsobit pro cílový systém. Jedná se o člena týmu, který má úplný přehled nad tokem dat ze zdroje, skrze migrační aplikaci a následně do cílového systému.

Technik Je člen týmu zodpovědný za přípravu migračního prostředí, průběh migrace a podávání informací všem zainteresovaným stranám.

Automat Je systém reprezentující všechny automatické činnosti uvnitř aplikace, které fungují na nějaký časovač. Typickým příkladem automatu je automatické spuštění migrační úlohy.

5.7 Doménový model

Na obrázku 5.2 je vyobrazený doménový model migrační aplikace. Aplikace vychází z existující migrační knihovny, takže byly přidány jen dva modely a to migrační úloha a časovač spuštění úlohy. Ostatní modely existovaly v migrační knihovně a aplikace s nimi pouze pracuje. Popis hlavních modelů aplikace:

Migrace Model reprezentující přesun dat z jednoho systému do druhého ve tvaru vhodném pro cílový systém. Nezáleží, jestli se zdrojový nebo cílový systém skládá z více komponent. Záleží na tom, aby se jednalo o ucelená data, která popisují celý objekt.

Migrační profil Konfigurační soubor DMT, který je reprezentován v XML formátu a popisuje všechny kroky migrace v aplikaci.

Migrační úloha Celek migrace reprezentující jeden či více migračních profilů. Migrační úlohu je možné automaticky nebo ručně spouštět, seskupovat podle ní záznamy z migrace a jinak odlišovat migrovaná data.

Záznam migrace Zaznamenaná informace o historii a stavu migrovaného objektu po operaci migračního nástroje. Může obsahovat informace o chybném, ale i validním zpracování. Záznam vzniká automaticky a jeho detail definuje nastavení logování migračního nástroje.

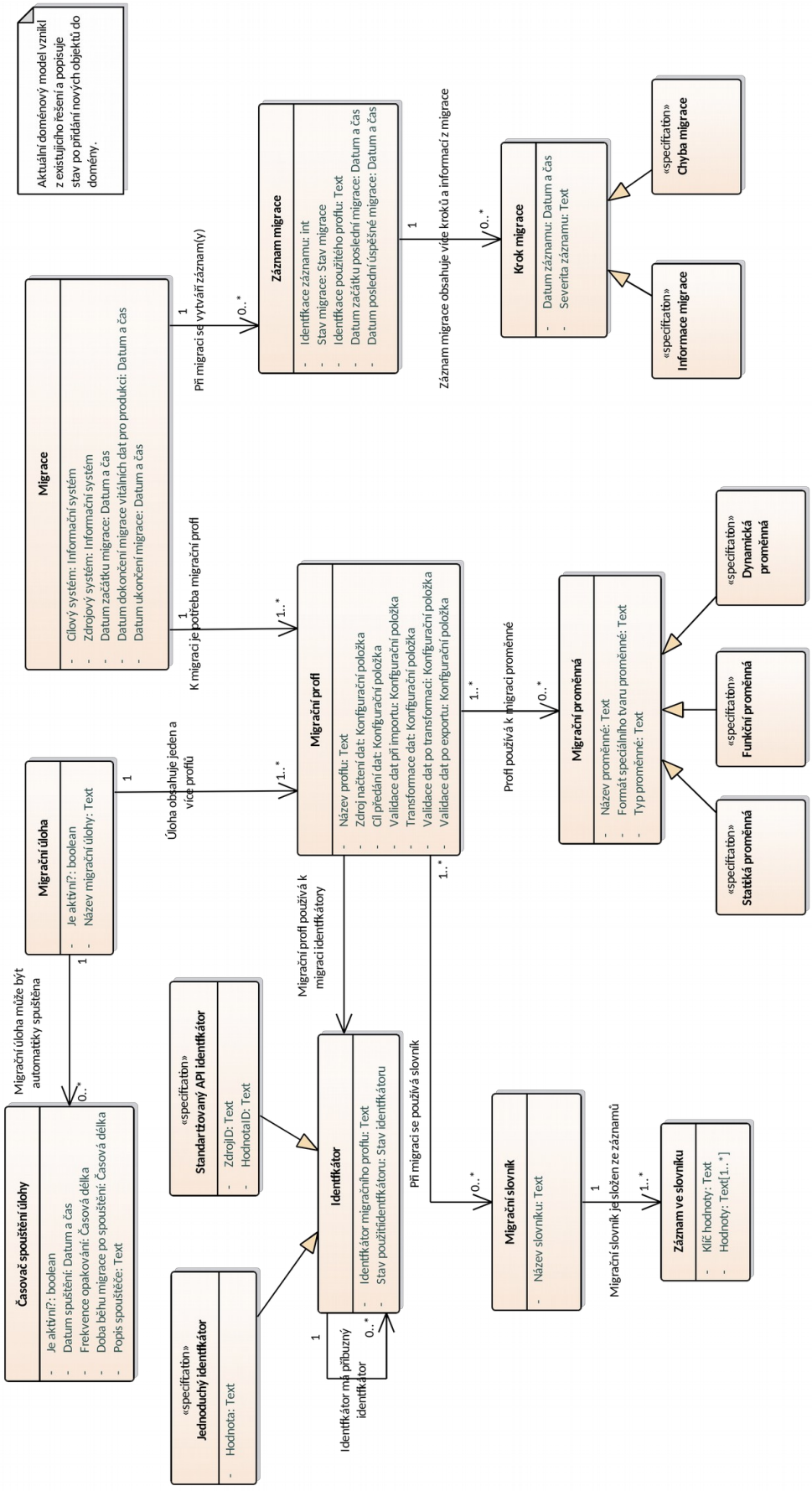
Migrační slovník V případě migračního slovníku se jedná o klasickou strukturu klíč–hodnota, která se používá v kontextu transformace a validace dat. Nejčastější použití je v případě potřeby přemapovat data z jednoho systému do druhého například při změně typu dokumentu, nebo přejmenování držitele dokumentu.

Migrační proměnná Model, který reprezentuje nějakou hodnotu uloženou v paměti migračního nástroje. Je možné ji definovat v migračním profilu a manipulovat s ní pomocí transformací. Též existují funkční proměnné, které jsou implementované funkce, ale pracuje se s nimi stejně jako s proměnnými. Například funkční proměnná `{@NOW}`, která vrátí aktuální čas.

Tabulka 5.1 Přehled realizace funkčních požadavků

	F01	F02	F04	F11	F12	F13	F14	F15	F21	F22
Programátor (fáze přípravy)										
UC100 – Příprava slovníků			x							
UC101 – Smazání slovníku			x							
UC102 – Vytvoření slovníku			x							
UC102a – Import slovníků do DMT			x							
UC103 – Úprava slovníku			x							
UC110 – Příprava migračního profilu	x									
UC111 – Vytvoření migračního profilu	x	x								
UC111a – Import migračního profilu	x									
UC112 – Úprava migračního profilu	x	x								
UC114 – Smazání profilu	x									
UC120 – Příprava nastavení k migraci	x									
UC121 – Export migračního profilu	x									
UC122 – Export slovníku			x							
Technik										
UC200 – Zastavení běhu migrace předčasně							x			
UC210 – Konfigurace DMT pro migraci				x		x				
UC211 – Přidání a úprava migrační úlohy				x						
UC212 – Přidání časovače na migrační úlohu						x				
UC230 – Ruční migrace dat				x	x					
UC231 – Zobrazení úloh migrací				x						
UC232 – Spuštění vybrané migrace					x					
UC240 – Zobrazení stavu migrací								x		
UC250 – Vyhodnocení stavu a chybovosti migrace								x	x	x
UC251 – Zobrazení/lokalizace chyb z migrace									x	
UC252 – Uložení logu z migrace do souboru										x
UC253 – Zobrazení statistiky										
Programátor (fáze dokončení)										
UC300 – Oprava nezdařené migrace	x		x						x	
UC310 – Migrace nezdařených dat					x					
Automat										
UC400 – Automatické spuštění migrace						x				
UC410 – Automatické zastavení migrační úlohy						x				

Obrázek 5.2 Doménový model migrační aplikace



Kapitola 6

Architektura a technologie

Jako technologický základ byly použity nástroje firmy Microsoft. Důvodem je, že existující knihovna je postavená na technologii Microsoft .NET Core 2.1. Dalším důvodem je, že autor práce a tým, který výslednou aplikaci bude udržovat, mají zkušenosti s touto technologií.

6.1 Třívrstvá neznamená tříúrovňová

„Tříúrovňová architektura je zavedená architektura softwarových aplikací, která organizuje aplikace do tří logických a fyzických výpočetních úrovní: prezentační úroveň, neboli uživatelské rozhraní, aplikační úroveň, kde se zpracovávají data, a datová úroveň, kde se ukládají a spravují data spojená s aplikací.“ [9]

Před vývojem všech aplikací je potřeba rozhodnout o jejich architektuře. V případě této aplikace byla použita klasická tříúrovňová architektura (viz definice výše). Z definice je jasně vidět na jaké části je potřeba tříúrovňovou aplikaci rozdělit. Jenomže, jak uznávají její autoři, tak může dojít k záměně mezi tříúrovňovou a třívrstvou architekturou „V diskusích o tříúrovňové architektuře se často používá zaměnitelné a chybné označení vrstva, jako například *prezentační vrstva* nebo *aplikační vrstva*.“ [9]. Proto je vhodné rozlišit mezi těmito dvěma architekturami. Rozdíl je, že vrstva označuje logický celek aplikace, kde rozdělení probíhá na úrovni zdrojového kódu, kdežto úroveň definuje rozdělení pomocí rozdílné infrastruktury, na které je aplikace spuštěna.

Typická tříúrovňová aplikace může vypadat následovně: grafické rozhraní, které běží na samostatném zařízení, jako je například mobilní telefon, nebo webová aplikace. Dále aplikační úroveň, kterou je webové rozhraní, se kterým komunikuje grafická aplikace. A poslední je databázový server jako Microsoft SQL Server, se kterým se komunikuje z aplikační úrovně. Existuje více důvodů k implementaci aplikace jako tříúrovňové. Z článku od IBM se jedná o [9]:

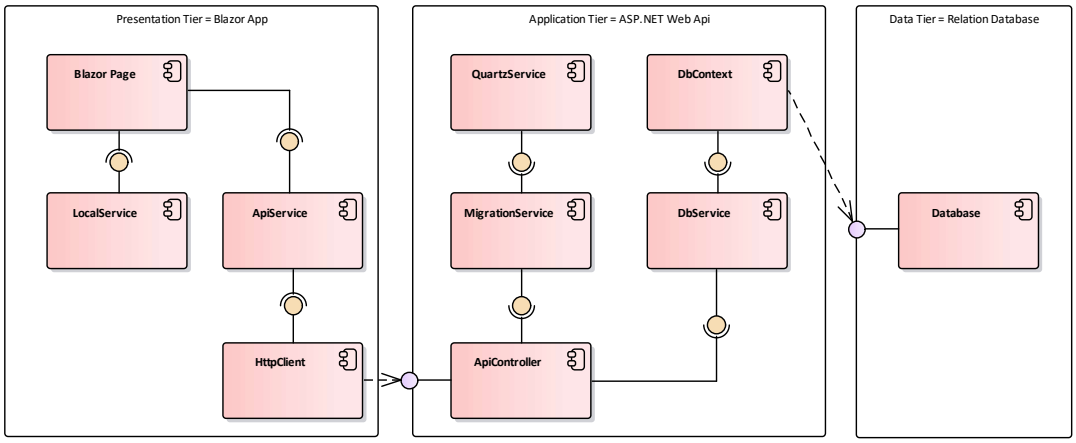
- rychlejší vývoj, protože na každé úrovni může pracovat jiný tým
- zjednodušenou škálovatelnost, protože je možné škálovat samostatné úrovně zvlášť
- lepší bezpečnost, protože uživatelské rozhraní nekomunikuje napřímo s databází, takže může aplikační úroveň provést potřebnou kontrolu a filtraci dotazů například od *SQL injection*

V případě vrstev se bavíme o podobném rozdělení: první vrstva je prezentační, tu vidí uživatel a pracuje s ní. Prostřední, aplikační vrstva, zajišťuje správné chování *business* logiky a zpracovává vstupně/výstupní požadavky z prezentační vrstvy. Poslední je datová vrstva, která obstarává manipulaci s daty, komunikaci s databází a persistenci dat.

Výsledná aplikace byla implementována jako tříúrovňová, ale z povahy jejího použití se nepředpokládá, že každá část aplikace bude pracovat v rozdílné infrastruktuře. Rozdělit jí tímto způsobem se s ohledem na problémy, které se při implementaci objevily viz sekce 7.1, zdálo přirozené.

Celá aplikace se skládá ze tří částí: webové aplikace vytvořené na technologii Blazor WebAssembly, webového rozhraní postavené na technologii ASP.NET Core WEB API a relační databáze, do které se ukládají všechna data za použití EF. Komunikace mezi webovým rozhraním a aplikací byla zajištěná pomocí REST rozhraní viz podsekce 6.4.5. Architektura aplikace je vidět na obrázku 6.1.

Obrázek 6.1 Architektura celé aplikace



6.2 .NET a .NET Core

V dokumentaci se píše „.NET je bezplatná vývojová platforma s otevřeným zdrojovým kódem pro vytváření mnoha druhů aplikací.“ [10]. Dále „.NET se vyskytuje v různých variantách, které se oficiálně nazývají implementace. .NET 5+ (včetně .NET Core) je nejnovější implementace a běží na jakékoli platformě. .NET Framework je původní implementace .NET a běží pouze v systému Windows.“ [10]. Termínem „na jakékoli platformě“ autor definice myslí pouze platformy, které jsou podporované. Důvodem této formulace je srovnání podpory platforem mezi původním .NET Framework a novým .NET Core, kterých je mnohem více.

Pro nové vývojáře bývá problém vyznat se ve verzích .NET a podporovaných platformách, proto je vhodné to nějakým způsobem objasnit. Zmatení nejspíše vychází z komplikovaného číslování a záměny významu slova. Z definice je vidět, že .NET je označení pro platformu a přidružené technologie, kdežto .NET Core a .NET 5+ je implementace .NET, která umožňuje běh na více prostředích. Je potřeba rozlišit, že .NET není stejné jako .NET 5+.

Původní byla první verze .NET Framework z roku 2002. V následujících letech vycházely její aktualizace. Po čtrnácti letech v roce 2016 vyšla první verze .NET Core, která nově podporovala multiplatformní běh. Zároveň ve stejném roce vyšla aktualizace původního .NET Framework ve verzi 4.6.2. Poslední verze .NET Framework 4.8 vyšla v roce 2019. Technologie .NET Core mezitím vyspěla do situace, kdy Microsoft přestal doporučovat původní .NET Framework pro nové projekty a všem vývojářům bylo doporučeno přejít na .NET Core. Od verze .NET 5 z roku 2020 je .NET Core známý pouze jako .NET.

Aktuální verze .NET 6 existuje ve variantě LTS. To znamená, že na rozdíl od předchozí verze, je podporovaná dva roky po jejím vydání. Aktuální model Microsoft je ten, že nové verze vychází každý rok, ale pouze každé dva roky vychází verze LTS. Verze, které nejsou LTS, jsou podporované zhruba rok. Podporou jsou myšleny opravy chyb a bezpečnostní záplaty. V případě aplikací, kde hraje velkou roli bezpečnost, může být potřeba pravidelně aktualizovat na nejnovější verzi .NET. Před použitím této technologie je nutné zvážit, zda jsou aktualizace každé dva roky v pořádku. Při aktualizaci na novější verzi může dojít k takzvaným *breaking changes*, nebo jiným komplikacím, které souvisí s aktualizací, proto je nutné brát to při výběru technologie v potaz.

6.3 Webová aplikace

Jak už bylo zmíněno, tak webová část aplikace je postavená na technologii Blazor WebAssembly. Blazor je unikátní technologie, protože umožňuje vytvořit SPA, česky jednostránkovou aplikaci, pouze za použití jazyků C# a HTML. Tato vlastnost přináší spoustu možností pro vývojáře, kteří nemají zkušenosti s jazykem JavaScript. Ten ve světě webových aplikací stále převládá – viz průzkum z předchozího roku na StackOverflow [11].

Na oficiálních stránkách pro WebAssembly je k nalezení definice „WebAssembly (zkráceně Wasm) je formát binárních instrukcí pro zásobníkový virtuální stroj. Wasm je navržen jako přenosný kompilační cíl pro programovací jazyky, který umožňuje nasazení klientských a serverových aplikací na webu.“ [12].

WebAssembly je spuštěn v prohlížeči uživatele a umožňuje spustit zdrojový kód, který je pro něj kompilovaný. Na rozdíl od JavaScript je potřeba zdrojový kód nejdříve kompilovat, kdežto JavaScript je interpretovaný za běhu. Z článku na Snipcart je možné najít srovnání JavaScript a Wasm [13] při použití ve webových aplikacích. Kompilace přináší jasné výhody: menší režie při spuštění aplikace, ověření typů používaných proměnných či objektů při kompilaci a rychlejší běh. Na druhou stranu Wasm má určité nevýhody: stále je v brzkém stádiu vývoje, takže pro běh určitých funkcí používá integraci s JavaScript, má problémy s bezpečností, kterou můžou útočníci využít a jedná se o komplexnější a složitější jazyk. Použití Blazor problém s komplexností odstraňuje, protože se při programování používá speciální Razor syntaxe viz sekce 6.3.2. Kód je pro WebAssembly kompilován, takže programátor nemusí znát WebAssembly syntaxi. Doporučení na konci zmiňovaného článku se srovnáním JavaScript a Wasm je, že použití Wasm je vhodné v případě aplikací náročných na výkon, což tato webová aplikace je.

SPA je označení pro jednostránkové webové aplikace, které běží v prohlížeči klienta místo na serveru. Název technologie vychází z toho, že aplikace je skutečně jednostránková a dynamičnost stránky je zajištěná neustálým přepisováním jejího obsahu při změně místo nového načtení celé stránky. Takové aplikace fungují spíše jako klasické *desktop* aplikace než webové stránky. Při spuštění musí být celá aplikace nejdříve stažena do prohlížeče uživatele. V tomto prohlížeči je následně spuštěna. Veškeré vykreslování stránek a komunikace s okolním světem probíhá z klientského prostředí. Komunikace se serverem probíhá pouze v případě potřeby získat data ze strany serveru, nebo při novém načtení celé stránky.

V dokumentaci Microsoft se probírají výhody použití SPA [14]. Konkrétně se jedná o podporu bohatého uživatelského rozhraní, rychlejší načítání dat, načítání dat na pozadí, svižnější uživatelské rozhraní a podpora uložení části vyplněného formuláře. Dále stojí za zmínku možná podpora běhu aplikace po výpadku připojení se serverem.

Nevýhodou použití SPA bývá delší doba prvního načtení stránky. Důvodem je potřeba načíst celou aplikaci při prvním načtení ze serveru. Existují různá řešení tohoto problému. Je možné provozovat hybridní řešení, kde aplikace pracuje ve dvou módech. První mód je Blazor Server, který běží do doby, než je celá aplikace stažena, poté se přepne a funguje jako Blazor WebAssembly. Pro potřeby této aplikace není nutné, aby se aplikace po prvním spuštění rychle načetla, tudíž není potřeba řešení implementovat.

6.3.1 Architektura webové aplikace

Webová aplikace je rozdělena na tři vrstvy: grafické rozhraní pomocí Razor pages na prezentační vrstvě, zpracování informací a vstupu uživatele pomocí služeb na aplikační vrstvě, persistence dat a komunikace s webovým rozhraním na datové vrstvě. Logické rozdělení aplikace ve zdrojovém kódu je zajištěné vrstvením aplikace. Z vytvořených Razor pages se přímo nevolá REST API, které poskytuje webové rozhraní. Místo toho jsou použity služby, které mají potřebné chování implementované. Tento vztah vyobrazuje obrázek 6.1. Technologie Razor pages je detailněji probrána v sekci 6.3.2.

Aplikace byla vytvořena použitím šablony pro generování projektů založených na knihovně MudBlazor. Oproti základní šabloně, která se používá standardně v případě vytvoření projektu přes Microsoft Visual Studio, je největším rozdílem přednastavení knihovny MudBlazor a použití vlastních prvků specifických pro tuto knihovnu. Prvky jsou myšleny třídy, které se používají při definici vzhledu stránky v Blazor. Jedná se například o prvek `Button`, který je nahrazený prvkem `MudButton`. Každý prvek podporovaný v Blazor má vlastní implementaci v MudBlazor. Tyto prvky poskytují snadno implementovatelné a kvalitně popsané rozhraní. Tyto prvky poskytují velké množství nastavení, které je pro potřeby aplikace možné vhodně upravit. Dále MudBlazor přináší nové zajímavé prvky, jako je například `MudTable`. `MudTable` je implementace dynamické tabulky, která umožňuje stránkování, řazení obsahu podle sloupců, možnost částečného načtení dat vlastním dotazem do databáze či rozhraní a mnoho dalšího. V neposlední řadě MudBlazor přináší velkou výhodu definicí vlastního a sjednoceného vzhledu všech prvků založeného na Material Design od firmy Google.

Služby ve webové aplikaci byly rozděleny na dva typy. Za první typ jsou považovány služby, které komunikují s webovým rozhraním. Každá ze služeb rozšiřuje abstraktní třídu `HttpService`, případně `HttpBulkService`, která implementuje základní CRUD operace. Tyto operace jsou definovány v rozhraní `ICrudContract`, případně `ICrudBulkContract`, které jsou blíže rozebrány v sekci 6.4.5. Toto rozhraní slouží především k definici kontraktu mezi webovou aplikací a webovým rozhraním.

Druhým typem jsou služby, které nekomunikují s webovým rozhraním. Konkrétně se jedná o: službu pro práci s uživatelskou schránkou, službu poskytující kódování textu podporované v aplikaci, službu pro validaci migračního profilu nebo službu pro práci s tabulkami. Poslední zmíněná služba je použita ve spojení se službou pro správu uživatelské schránky. Díky tomu může uživatel exportovat data z tabulek v aplikaci do schránky a přenést do aplikace Excel nebo do jiné tabulkové aplikace, která podporuje stejný formát.

Každá ze služeb implementuje rozhraní, které definuje jejich veřejně viditelné vlastnosti a metody. Použitím těchto rozhraní je možné části kódu jednodušeji testovat pomocí *unit testing* (jednotkové testování) viz kapitola 9. Dalším důvodem k použití *interface* je podpora DI, česky vkládání závislostí, které Blazor standardně umožňuje. Veškerá konfigurace, nejenom pro DI, probíhá při spuštění aplikace ve třídě `Program`.

6.3.2 Blazor a Razor

Definice z dokumentace „Blazor je framework pro vytváření interaktivního webového uživatelského rozhraní na straně klienta pomocí rozhraní .NET.“[15].

Blazor dovoluje vytvářet dynamické UI prvky bez nutnosti použití jazyka JavaScript. Tyto prvky se skládají z kombinace jazyka C# a značek HTML ve struktuře podobné XML, které se říká Razor syntaxe. V případě Blazor se jedná o evoluci dřívější technologie Razor pages. Těmto UI prvkům se v kontextu Blazor říká komponenty. Použitím komponent je možné seskupit chování a vzhled pod jeden zdrojový kód podobně jako tomu je při použití tříd. Při vytváření konkrétní stránky může programátor pracovat s vytvořenými komponentami a postupným skládáním sestavit webovou stránku. Například v této aplikaci byly použity komponenty pro tabulky, tlačítka, formuláře a dialogy. Na útržku kódu 6.1 je vidět použití technologie Blazor.

Jedná se o implementaci komponenty `YesNoDialog` za použití prvku `MudDialog`. Tento dialog slouží k získání ano/ne odpovědi od uživatele. Je možné mu předat otázku, barvy a ikony tlačítek.

Výpis kódu 6.1 Implementace komponenty YesNoDialog

```
<MudDialog>
    <DialogContent>@DialogMessage</DialogContent>
    <DialogActions>
        <MudButton ...>Ne</MudButton>
        <MudButton OnClick="Submit" ...>Ano</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [Parameter]
    public string DialogMessage { get; set; }

    ...
    void Submit()
    {
        MudDialog.Close(DialogResult.Ok(true));
    }
    ...
}
```

6.3.3 Použití JavaScript

Při implementaci Blazor aplikace není nutné používat jazyk JavaScript. Velká část funkcí je vytvořena, případně existuje mnoho podpůrných knihoven. Nicméně, v případě potřeby, technologie Blazor nebrání použití jazyka JavaScript. Tato svoboda je při vytváření webové aplikace neocenitelná. Je možné implementovat všechny funkce, které Blazor nepodporuje takzvaně „out of the box“ za použití JavaScript a API prohlížeče. V aplikaci byly tímto způsobem implementovány následující funkce: ukládání souborů pomocí ukládacího dialogu s výběrem umístění, kopírování textu do uživatelské schránky a manipulace s lokálním úložištěm prohlížeče pro ukládání nastavení.

Implementace služby pro správu uživatelské schrány vypadala následovně: byla vytvořena obslužná instanční třída s veřejným rozhraním, do které se pomocí konstruktoru předává *JavaScript runtime* viz útržek kódu 6.2.

Výpis kódu 6.2 ClipboardService.cs

```
public class ClipboardService
{
        private readonly IJSRuntime _jSRuntime;

        public ClipboardService(IJSRuntime jSRuntime)
        {
                _jSRuntime = jSRuntime;
        }

        public async Task CopyTextToClipboard(string textToCopy)
        {
            await _jSRuntime.InvokeVoidAsync("CopyText", textToCopy);
        }
}
```

Následně byla vytvořena samostatná funkce v jazyce JavaScript ve zdrojovém souboru viz útržek 6.3, který byl vložen do složky „wwwroot/javascript“.

Výpis kódu 6.3 Clipboard.js

```
function CopyText(text) {
    navigator.clipboard.writeText(text)
}
```

Nakonec byl přidán odkaz na zdrojový soubor `Clipboard.js` do souboru `index.html` viz útržek 6.4. Tím bylo aplikaci řečeno, odkud má funkci načíst při načtení webové aplikace.

Výpis kódu 6.4 index.html

```
<!DOCTYPE html>
<html>
...
<body>
...
<script src="javascript/Clipboard.js"></script>
...
</body>
</html>
```

Velkou výhodou při implementaci JavaScript pomocí služeb je přidání rozhraní s pevně definovanými typy, které je v aplikaci k nerozpoznání od jiných služeb. To znamená, že je možné použít JavaScript stejným způsobem jako jiné služby. Navíc s výhodou validace předávaných parametrů při kompilaci.

6.4 Webové rozhraní

Webové rozhraní slouží jako prostředník mezi webovou aplikací a migrační knihovnou. Rozhraní bylo vytvořeno pro potřeby přístupu ke zdrojům serveru, na kterém běží migrace. Důvodem je, že není možné, aby migrace probíhala v prohlížeči uživatele viz 7.2.

Rozhraní je postavené na technologii ASP.NET Core WEB API. Jedná se o framework, který umožňuje vyvíjet webová API, ke kterým můžou přistupovat z různých míst různí klienti včetně mobilních aplikací, webových prohlížečů, nebo klasických aplikací pomocí HTTP.

6.4.1 Architektura webového rozhraní

Architektura webového rozhraní je velmi podobná architektuře webového aplikace. Obě aplikace jsou třívrstvé a tyto vrstvy splňují podobné účely, obě aplikace používají k implementaci logiky služby a obě aplikace v jádru používají DI, které konfiguruji stejným způsobem při startu aplikace.

V aplikaci byly implementovány tyto vrstvy: prezentační vrstva reprezentována kontrolery[1], nebo též API *endpoints*, aplikační vrstva, která se skládá z obslužných služeb a datová vrstva, která je implementována za použití technologie Entity Framework Core. Aplikace byla rozdělená do vrstev za použití logického oddělení zdrojových kódu. Na obrázku 6.1 je možné blíže sledovat architekturu rozhraní.

Kontroler nekomunikuje napřímo s databází, ale místo toho používá službu, která implementuje veškeré potřebné metody. Všechny kontrolery implementují rozhraní `ICrudContract`, případně `ICrudBulkContract`, které přesně definují, jaké *endpoints* webová aplikace očekává. Kontrakty jsou blíže popsány v sekci 6.4.5.

[1] Kontroler je překlad anglického *controller*, které se používá v oficiální české verzi dokumentace pro .NET

Služba může, ale nemusí pracovat s databázovou vrstvou. Například služba `QuartzService` nabízí možnost plánování úloh v aplikaci pomocí knihovny Quartz.NET viz sekce 6.4.2, ale nevyužívá k tomu přístup do databáze. Pokud služba pracuje s databází, tak k tomu používá DAL místo přímého přístupu viz sekce 6.5.

Použitím DAL bylo docíleno odstínění implementace přístupu do databáze od aplikace. Tudíž není potřeba na úrovni aplikační vrstvy volat SQL dotazy nad databází, ale pouze použít rozhraní vystavené DAL.

6.4.2 Quartz.NET

Z oficiálních webových stránek Quartz.NET je možné vyčíst definici „Quartz.NET je plnohodnotný open source systém pro plánování úloh, který lze použít od nejmenších aplikací až po rozsáhlé podnikové systémy.“ [16]. Jedná se o jeden z klasických způsobů jakým se v .NET plánují dlouhotrvající a opakovatelné úlohy.

Systém byl implementován pomocí služby `QuartzService`, která nabízí všechny potřebné metody pro její operování viz ukázka rozhraní 6.5. Z ukázky je vidět, že se jedná o asynchronní službu.

Výpis kódu 6.5 Rozhraní IQuartzService

```
public interface IQuartzService
{
    Task<bool> IsJobRunningAsync(JobKey jobKey);
    Task<bool> IsJobScheduledAsync(JobKey jobKey);
    Task<IEnumerable<TriggerKey>> ScheduleJobAsync(...);
    Task<IEnumerable<TriggerKey>> StartJobAsync(...);
    Task StopJobAsync(JobKey jobKey);
    Task UnscheduleJobAsync(JobKey jobKey);
}
```

Služba je při startu aplikace konfigurována v kontextu DI jako *singleton*. To je dáno tím, že třída v níž probíhá zpracování úloh je vytvářena v každé instanci třídy `QuartzService` zvlášť. Tudíž by jejím zánikem zanikly i běžící úlohy.

Knihovna Quartz.NET vystavuje veřejné rozhraní `IJob`, které musí použít každá třída implementovaná jako úloha. Toto rozhraní definuje pouze jedinou metodu viz útržek kódu 6.6.

Výpis kódu 6.6 Rozhraní IJob

```
public interface IJob
{
    Task Execute(IJobExecutionContext context);
}
```

6.4.3 Použití migrační knihovny

Kritickou částí aplikace je propojení s migrační knihovnou. Aby toto propojení bylo co nejefektivnější, byla migrační knihovna částečně upravena. Jednalo se především o tyto změny:

Změna logování Místo statické třídy implementující rozhraní `ILogger` byla přidána statická třída `DMTLogger` zodpovědná za vytvoření instancí logovacích tříd, se kterými pracují všechny třídy v migrační knihovně. Dále byl nahrazen logovací *interface* z knihovny log4net za *interface* z oficiálních knihoven Microsoft. Logování stále probíhá za pomoci knihovny log4net, ale nyní se používá v kombinaci s rozšiřující knihovnou pro logování od Microsoft. Konkrétní název knihovny je `Microsoft.Extensions.Logging.Log4Net.AspNetCore`. Tím bylo umožněno konfigurovat logování migrační knihovny z aplikace, která jí používá. Výhodou je použití

logování, které se standardně používá v aplikacích postavených na ASP.NET. Do změny se migrační knihovna konfigurovala sama při inicializaci.

Změna nastavení Veškeré statické nastavení bylo přesunuto do instanční třídy `AppSettings`, která se inicializuje při startu aplikace jako součást statické třídy `GlobalSettings`. Toto nastavení je možné změnit externě předáním vlastní instance `AppSettings`, nebo předáním sekce z konfigurace typu `IConfigurationSection`. Aplikace z této konfigurace poté sestaví instanci `AppSettings`. V obou případech se nově validuje stav nastavení pomocí `DataAnnotations` definovaných ve zmíněné třídě s nastavením.

Změna v databázové vrstvě Další změnou bylo odtržení DAL z migrační knihovny a přesunutí do vlastní databázové knihovny. Při této změně proběhly úpravy nad metodami, které pracovaly s větším množstvím dat. Nově byly použity funkce pro hromadné zpracování pomocí knihovny `EFCore.BulkExtensions`. To znamená, že se zkrátily doby nahrávání a selekce dat. Největší rozdíl byl zaznamenám při nahrávání slovníků do knihovny ze souboru.

Změna verze Poslední změna, která v migrační knihovně proběhla, byla povýšení z verze .NET Core 2.1 na verzi .NET 5.

Po těchto změnách bylo možné migrační aplikaci a migrační knihovnu jednoduše propojit. K tomu posloužila třída `MigrationJob`, která implementuje zmíněné rozhraní `IJob` z knihovny Quartz.NET. Při konfiguraci migrační úlohy jsou všechna nastavení předána pomocí vlastní třídy `MigrationJobData`. Při spuštění úlohy se provede rozbalení všech nastavení, uloží se záznam o spuštění migrace do databáze a potom se migrace spustí.

6.4.4 REST

„REpresentational State Transfer je architektonický styl pro poskytování standardů mezi počítačovými systémy na webu, který usnadňuje vzájemnou komunikaci systémů. Systémy kompatibilní s REST, často nazývané RESTful, se vyznačují tím, že jsou bezstavové a oddělují zájmy klienta a serveru.“ [17].

Webová aplikace a webové rozhraní jsou propojené pomocí rozhraní, které implementuje REST. REST se dívá na pojem rozhraní zdrojově. K datům se přistupuje pomocí URL cesty v kombinaci s identifikátorem zdroje. Metoda manipulace s daty je odlišena pomocí použité HTTP metody a předaného zdroje. REST v základní implementaci umožňuje přistupovat k datům pomocí čtyř různých metod:

GET metoda pro získání zdroje

Výpis kódu 6.7 Metoda GET

```
GET /api/job
GET /api/job/qaKGnrDcdbA
```

POST metoda pro vytvoření zdroje

Výpis kódu 6.8 Metoda POST

```
POST /api/job
{ name="Úloha" }
```

PUT metoda pro přepsání, či vytvoření zdroje

Výpis kódu 6.9 Metoda PUT

```
PUT /api/job/qaKGnrDcdbA
{ name="Úloha 1" }
```

DELETE metoda pro smazání zdroje

Výpis kódu 6.10 Metoda DELETE

```
DELETE /api/job/qaKGnrDcdbA
```

Za zmínku stojí ještě metoda `PATCH`, která se při použití REST rozhraní také implementuje. Tato metoda slouží k aktualizaci vybrané části zdroje. Při jejím použití je potřeba definovat URL cestu ke konkrétní položce konkrétního zdroje a tu jako jedinou aktualizovat. Na rozdíl od metody `PUT` provádí pouze aktualizaci, kdežto metoda `PUT` úplně přepisuje zdroj nově předaným zdrojem. Metoda `PATCH` není ve webovém rozhraní implementována.

6.4.5 Propojení rozhraní a aplikace

Propojením jsou myšleny veškeré systémy, které byly vytvořeny pro komunikaci mezi webovou aplikací a webovým rozhraním.

Implementace aplikace i rozhraní ve stejném jazyce umožnila sdílet všechny společné zdrojové kódy v jedné společné knihovně. Proto byly veškeré společné zdroje jako jsou DTO, *interfaces* nebo *extensions*, umístěny do společné knihovny, která byla přidána do obou aplikací. Výhodou tohoto přístupu je možnost definice veškerých modelů pouze jednou. V případě jakékoli úpravy je změna reflektována v obou aplikacích najednou.

Na těchto základech bylo vybudováno zajímavé propojení, které autor aplikace nazývá kontrakty. Kontrakt je v případě této aplikace *interface*, který definuje operace nad konkrétním DTO s vybraným typem identifikátoru. V základu se jedná o klasické CRUD operace, které jsou podle potřeby rozšířeny o další operace. Všechny tyto operace jsou definovány s ohledem na architekturu REST, takže je možné je implementovat pomocí HTTP metod zmíněných v sekci 6.4.4. Základem těchto kontraktů jsou dvě rozhraní. První a nejpoužívanější je `ICrudContract`. Druhý kontrakt je `ICrudBulkContract`, který se liší pouze tím, že definuje metody pro manipulaci s hromadným množstvím dat. Konkrétní kontrakt se může skládat z více rozhraní. Pro definici typu DTO, se kterým se v kontraktu pracuje a typu identifikátoru daného DTO jsou použity generiky viz 6.11.

Ve webové aplikaci slouží kontrakt jako základ pro službu, která se používá na úrovni Razor pages. K této implementaci byly vytvořeny abstraktní třídy `HttpService` a `HttpBulkService`, které dědí ze základních kontraktů `ICrudContract` a `ICrudBulkContract`. Tyto třídy implementují všechny operace definované v základních kontraktech. Implementací je myšleno, že se pomocí `HttpClient` provolávají koncové body rozhraní, připravují parametry před voláním a získávají odpovědi, které pak předávají dále. Obrovskou výhodou tohoto přístupu je sjednocené chování pro všechny základní operace. Konkrétní kontrakty jsou poté implementovány pomocí stejnojmenných služeb.

Ve webovém rozhraní je implementována druhá strana kontraktu. Každý kontroler dědí přímo z třídy `ControllerBase` a konkrétního kontraktu. Veškeré operace jsou vytvořeny podle předlohy z kontraktu. Pomocí atributů jsou metody označeny příslušnými HTTP metodami. Tyto kontrolery obstarávají základní funkce tak, že předají veškeré dotazy dál do obslužných služeb a vrací odpovědi. Odpovědí může být vytvořený či aktualizovaný objekt, klasický „OK“ status operace, nebo chyba.

Veškeré operace všech kontraktů byly definované jako asynchronní. Důvodem je především vynucení asynchronnosti na obou stranách kontraktu. V takovém případě aplikace může uživateli při používání přijít svižnější, protože při volání implementovaných operací není blokováno

Výpis kódu 6.11 Kontrakt pro migrační úlohu

```
public interface ICrudContract<TIdentifier, TType>
{
    Task<IEnumerable<TType>> GetAll(SearchDto search = null);
    Task<TType> Get(TIdentifier identifier);
    Task<TType> Add(TType item);
    Task Update(TIdentifier identifier, TType item);
    Task Delete(TIdentifier identifier);
}

public interface IJobContract : ICrudContract<string, JobDto>
{
    Task StartJob(string jobId);
    Task StopJob(string jobId);
    Task EnableJob(string jobId);
    Task DisableJob(string jobId);
}
```

UI vlákno. To znamená, že stále dochází k překreslování uživatelského rozhraní jako reakce na vstup uživatele. Webové rozhraní může být v některých situacích rychlejší viz článek na webu Gokhansengun [18].

6.5 Databázová vrstva a modely

Databázová vrstva, nebo zkráceně DAL, skrývá všechny technologie, které jsou spojené s ukládáním dat do databáze. Vrstva slouží jako abstrakce pro manipulaci s databází, protože implementace je za touto vrstvou schovaná. V praxi programátor pracuje pouze s třídou, nebo knihovnou, která obstarává veškerou komunikaci s databází. Může se jednat o jednoduchou souborovou databázi reprezentovanou jako excelový soubor, ke kterému se přistupuje pomocí ODBC ovladače pomocí `OdbcConnection`. Případně složitější relační databázi jako je například Microsoft SQL Server, ke které se přistupuje pomocí EF.

Veškerá komunikace s databází byla zajištěna pomocí technologie Entity Framework Core, zkráceně EF Core, ve variantě *code first*. *Code first* znamená, že jsou nejdříve navrženy všechny entity pomocí tříd ve zdrojovém kódu. EF je pak schopný z těchto entit vytvořit celou databázi. Jedná se o opak *database first* přístupu, ve kterém jsou entity generovány z už existující databáze. Entity Framework implementuje techniku známou jako ORM. To je automatické mapování objektů, též známých jako entity, na tabulky relačních databází. Mapování je možné definovat a upravit podle potřeby viz ukázka kódu 6.12. EF Core podporuje širokou škálu relačních databází.

V kontextu vyvíjené aplikace byly všechny zdrojové kódy, spojené s databázovou vrstvou, umístěné do vlastní knihovny. Tím bylo zajištěno fyzické oddělení DAL od webového rozhraní. DAL není závislý na žádné další knihovně/aplikaci, ale ostatní aplikace jsou závislé na něm.

V aplikaci databázovou vrstvu reprezentuje třída `AppDbContext`, která vystavuje tři *property* (položky) typu `DbSet`. Jedná se o položky, které představují přístup tabulkám, reprezentované jako entity typu `JobEntity`, `TriggerEntity` a `ProfileEntity`. Strukturu nové databáze určené pouze pro migrační aplikaci je možné vidět na obrázku 6.2. S třídou `AppDbContext` pracují služby z webového rozhraní, kde zajišťují základní CRUD operace.

Výpis kódu 6.12 Třída TriggerEntity

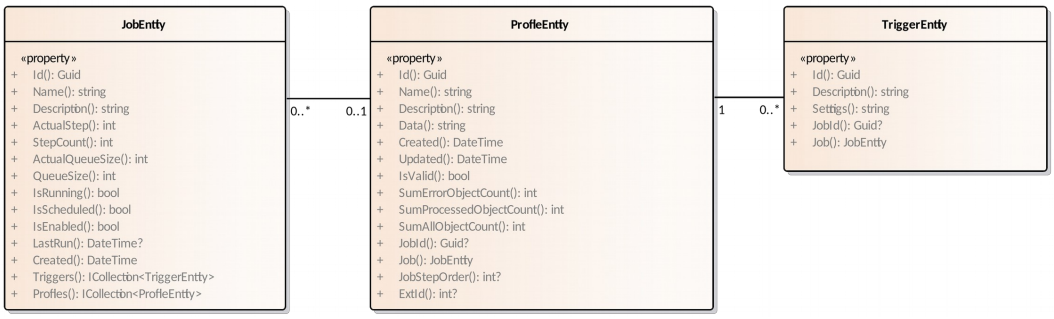
```
public class TriggerEntity
{
public Guid Id { get; set; }
public string Description { get; set; }

public class TriggerConfiguration : IEntityTypeConfiguration<TriggerEntity>
{
    public void Configure(EntityTypeBuilder<TriggerEntity> builder)
    {
        // Vlastnosti tabulky, indexy a FK
        builder.HasKey(ent => ent.Id);

        // Vlasntosti konkretnich sloupecku
        builder.Property(ent => ent.Id)
            .ValueGeneratedOnAdd();

        builder.Property(ent => ent.Description)
            .HasMaxLength(1024);
    }
}
}
```

Obrázek 6.2 Nové databázové entity



Kapitola 7

Postup implementace

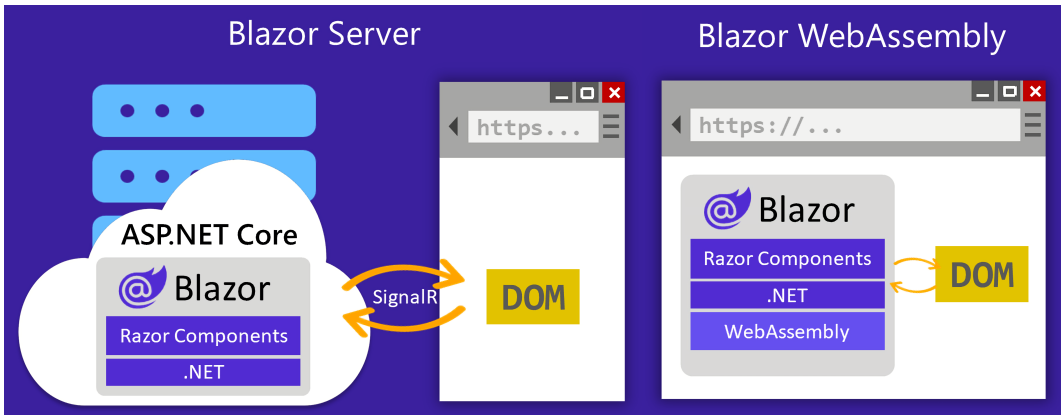
7.1 Implementace webové aplikace

Nejdříve byla vytvořena webová aplikace založená na technologii Blazor Server se samostatnou SQLite databází. Jednalo se o jednoduchou aplikaci se základním textovým editorem, který umožňoval zvýraznit klíčová slova a strukturu obyčejného XML dokumentu.

V této fázi byl objeven první nedostatek vybrané technologie. Tím byla pomalá aktualizace *syntax highlight* (zvýraznění syntaxe) klíčových slov. Problém se projevoval při editaci dokumentu, který měl více něž několik desítek řádků. Pro uvedení v kontextu migračních profilů se může jednat až o stovky řádků konfiguračního souboru. Při kontrole migračního profilu docházelo k několika vteřinovému zamrznutí aplikace z důvodu potřeby předat celý dokument na server, provést jeho kontrolu a poté vrátit klientovi a znovu vykreslit se zvýrazněním. Funkce zvýraznění klíčových slov slouží pro lepší orientaci ve struktuře upravovaného migračního profilu. Tudíž se jedná o klíčovou funkci, která musí být součástí editoru.

Z toho důvodu došlo k přechodu z Blazor Server na technologii Blazor WebAssembly, která na rozdíl od původní technologie běží v prohlížeči uživatele viz dokumentace na stránkách Microsoft [15]. Na obrázku 7.1 je vidět vizuální srovnání obou technologií. Tato změna stačila k tomu, aby se úplně odstranil problém s rychlostí. Důvodem je, že WebAssembly nepotřebuje při kontrole dokumentu komunikovat se serverem, tudíž neprobíhá žádné předávání dat.

Obrázek 7.1 Blazor Server a WebAssembly, převzato z `https://docs.microsoft.com/`



7.2 Přidání webového rozhraní

Po přepsání aplikace za použití Blazor WebAssembly se objevil nový problém, který vycházel z toho, že aplikace běží v prohlížeči uživatele, nikoli na serveru. Aplikace neměla přístup ke zdrojům serveru. To se týkalo už zmíněné databáze, nebo složek systému, které se používají při migraci. Proto byla vytvořena druhá aplikace, která umožňuje přímo používat zdroje serveru. Jednalo se o webové rozhraní založené na technologii ASP.NET Web API.

Obě aplikace byly vzájemně propojené pomocí REST rozhraní. Toto rozhraní bylo implementováno podle *best practices* (doporučovaných praktik) popsaných v článku na StackOverflow [19]. To znamená, že byly přidány funkce filtrace, stránkování, nebo řazení obsahu. Pro komunikaci s webovým rozhraním byla použitá knihovna Polly, která zajišťuje opakované volání cílového rozhraní v případě chybných odpovědí. Tím byla přidána jednoduchá konfigurace opakování dotazů při jejich selhání, například z důvodu *timeout* (vypršení časového limitu).

Následně byly přidány základní funkcionality pro otestování minimálního produktu. Jednalo se o funkce editace migračního profilu, nastavení migračních úloh a spuštění migrace na serveru přes webové rozhraní.

7.3 Přidání nových funkcí

Po testech funkčnosti technologie a návrhu aplikace byly implementovány zbylé funkce. Konkrétně se jednalo o funkce pro vytvoření a editaci migračního slovníku, funkce pro zobrazení informačních záznamů z migrace a základní statistika zpracování migrace.

Většina funkcí byla implementována tím způsobem, že byla na straně serveru vytvořena entita, která reprezentovala data v databázi za použití ORM nástroje EF Core. Poté byla vytvořena obslužná třída, která obstarává *business* logiku a pracuje s databázovou vrstvou. Následně byl vytvořený *interface*, reprezentující kontrakt mezi klientskou aplikací a serverem. Tento kontrakt byl pak použit při implementaci REST *endpoint* na straně serveru a HTTP služby na straně klienta. Dále byla implementována konkrétní služba pro danou entitu na straně klienta, která pracovala s HTTP službou. A jako poslední byla v klientské aplikaci vytvořena stránka, která uživateli reprezentovala potřebné informace a umožňovala pracovat s prostředky serveru s ohledem na danou entitu.

7.4 Implementace grafického rozhraní

V této fázi byly rozpracované základní stránky editoru migračního profilu, plánování úloh, ale ostatní stránky chyběly.

7.4.1 Zlatá pravidla

Pro zajištění kvality byla při tvorbě aplikace použita existující pravidla pro tvorbu uživatelského rozhraní. V knize *Designing the User Interface: Strategies for Effective Human-Computer Interaction* se píše o osmi zlatých pravidlech, která je vhodné dodržovat [20, sekce 3.3.4]. Čtyři nejdůležitější s ohledem na vytvářenou aplikaci jsou:

1. být konzistentní
2. nabídnout informativní zpětnou vazbu
3. navrhovat dialogy vedoucí ke zdárnému konci
4. předcházet chybám

Na konzistenci je možné dívat se ze dvou úhlů: vzhled aplikace a chování aplikace při běhu. Konzistence vzhledu byla zajištěná použitím webové knihovny MudBlazor, která vychází z existujícího *Material Design* od firmy Google. Z dokumentace pro *Material Design* je možné vyčíst definici „Material je designový systém vytvořený společností Google, který pomáhá týmům vytvářet vysoce kvalitní digitální zážitky pro Android, iOS, Flutter a web.“ [21]. Knihovna MudBlazor zajišťuje jednotný vzhled všech stránek pomocí předem definovaného vzhledu ovládacích prvků od jednoduchých tlačítek po složité tabulky. Konzistence chování byla dosažena implementací jednotného chování pro širokou škálu akcí. Všechna důležitá data jsou zobrazená v podobných tabulkách s možností procházet stránkovaný obsah. Při editaci hodnot vyskočí podobná modální okna ve kterých uživatel může upravit informace. Tímto konzistentním chováním se autor pokusil usnadnit cílovým uživatelům orientaci v aplikaci.

Zpětná vazba je uživateli předávaná více způsoby. V případě vyplňování formulářů při editaci nějakého modelu jsou zobrazovány informace o chybách vedle vyplňovaných políček. Takový chybný formulář není možné odeslat ani uložit. Po úspěšné aktualizaci objektu, nebo po spuštění či zastavení migrace, jsou uživateli zobrazovány informace pomocí prvku `SnackBar`. Informace jsou zobrazovány pomocí zpráv, které jsou umístěné v pravém spodním rohu stránky a po několika sekundách zmizí. V případně nezvratných chyb je uživateli zobrazena informace pomocí modálního okna, kterou je potřeba ručně potvrdit.

V aplikaci byly vytvořeny pouze jednostránkové dialogy, které neovlivňují více, než se od nich logicky očekává. Například, dialog pro přejmenování migračního profilu provede pouze přejmenování migračního profilu.

Chybám bylo předcházeno pomocí kontroly stavu objektu při zpracování, vypínání částí grafického rozhraní, pokud by použití vedlo k chybě a dalšími způsoby. Například není možné migrační úlohu měnit, pokud je aktivní. Pokud by uložení dialogového okna vedlo k chybě, tak není možné jej uložit. Dále se provádí kontrola uživatelem zadávaných dat ve vyplňovaném formuláři a po odeslání se provádí kontrola i ve webovém rozhraní.

7.4.2 Inspirace

Inspirací při tvorbě stránek byly již existující aplikace. Pro správu úloh migrace posloužila aplikace *task manager* (plánovač úloh) operačního systému Windows. Při tvorbě editoru migračního profilu posloužila klasická představa editoru zdrojového kódu. Konkrétní inspirace vycházela z webové aplikace XML–EDITOR[1]. Inspirací z již existujících aplikací si autor sliboval, že uživatel bude mít snadnější orientaci v této aplikaci.

7.4.3 Implementace migračního editoru

V případě implementace stránky s migračním editorem byla použitá technologie Monaco Editor. Jedná se o open–source editor zdrojového kódu od firmy Microsoft, který je použitý například ve známé aplikaci Visual Studio Code, též od společnosti Microsoft.

Monaco Editor poskytuje aplikaci základní funkcionality, jako je zvýraznění syntaxe pro XML dokumenty. Dále umožňuje uživateli vyhledávat v textu, nahrazovat text, vracet se o krok zpět po editaci, nebo jít o krok dopředu. Podle potřeby je možné editor konfigurovat pomocí *JavaScript*, nebo rozšířit o podporu vlastního jazyka. Aplikace se napojuje na *events* (události) Monaco Editor a při každé změně obstarává základní logiku. Tou může být validace profilu, nebo automatické vyplňování textace.

Pro validaci migračního profilu byla vytvořena služba `ProfileValidatorService`, která poskytuje všechny potřebné funkce pro validaci profilu. Tato služba se používá pouze ve spojení s editorem migračního profilu. Rozhraní této služby je možné vidět níže ve výpisu kódu 7.1.

[1] Aplikace XML–EDITOR je dostupná z `https://jsonformatter.org/xml-editor`

Výpis kódu 7.1 Rozhraní IProfileValidatorService

```
interface IProfileValidatorService
{
    void AddValidation(IElementValidatorAction action);
    void AddValidation(IAttributeValidatorAction action);
    bool CheckProfileXmlIsValid(string profileAsXml, out ...);
    List<ValidatorWarning> ValidateProfile(string profileAsXml);
    ...
}
```

Třídu je možné konfigurovat pomocí metod `AddValidation`. Metodám se předají třídy pro validaci elementu, nebo atributu migračního profilu. Tyto třídy musí implementovat k tomu určená rozhraní. Aktuálně jsou implementovány dvě kontroly: kontrola existence slovníku a kontrola pro existenci proměnné. Pro ověření správnosti profilu pomocí těchto kontrol pak slouží metoda `ValidateProfile`, která v případě problému vrátí seznam chybových elementů/atributů.

Pro ověření správnosti tvaru migračního profilu slouží metoda `CheckProfileXmlIsValid`. Ta v případě chybějících, nebo přebývajících položek v migračním profilu vrátí seznam upozornění pomocí `out` argumentu. Pokud se při validaci objeví větší problém, jako je například nevalidní XML dokument, tak metoda `CheckProfileXmlIsValid()` vrátí příznak o chybě.

7.4.4 Implementace tabulek

Tabulky jsou použity na většině stránek aplikace, tudíž bylo potřeba zajistit, aby nebylo zbytečně problematické s nimi pracovat. Proto byly tabulky implementovány podle pravidel implementace tabulek v podnikovém prostředí viz článek [22]. V daném článku se píše o základních funkcích, které by měly tabulky nabízet. V případě aplikace byly implementovány pouze funkce, které dávaly smysl, jedná se o:

- vyhledávání obsahu
- komplexnější filtrace obsahu
- řazení dat podle sloupečku
- editace dat
- stránkování a listování daty
- více akcí nad tabulkou, například přidání nového záznamu
- více akcí nad řádkem, například smazání/editace vybraného záznamu
- export dat z tabulky

Neplatí, že by byly všechny funkce implementovány pro každou tabulku. Funkce pro složitější filtraci obsahu byla použita pouze u tabulek, ve kterých se očekávalo velké množství dat k filtraci. Jednalo se především o tabulku pro zobrazení všech chyb při migraci. Zde bylo očekáváno několik miliónů záznamů na migraci. Každá tabulka sice umožňuje filtrovat data při zobrazení uživateli, ale pouze v případě zobrazení chyb z migrace se jedná o plnohodnotný filtr. V ostatních tabulkách se podle seznamu slov, vyplněných v pravém horním rohu tabulky, provádí vyhledávání ve všech sloupečcích a všech řádcích dat. To znamená, že filtrace probíhá na straně uživatele. Pokud je nějaké z hledaných slov nalezeno, tak se uživateli zobrazí v tabulce řádek s tímto údajem.

Protože tabulka se záznamy chyb z migrace může obsahovat miliony záznamů, tak bylo implementováno jednoduché stránkování. Při dotazování webového rozhraní je možné předat informaci

od jaké pozice má stránka začínat a jaká je velikost požadované stránky. V tabulce je možné použít tuto vlastnost a zobrazovat uživateli pouze tento „výřez“. V případě tabulek pro seznam migračních profilů, nebo slovníků, jsou data stránkována až v prohlížeči.

Exportovat data je možné pouze v případě tabulky pro zobrazení všech chyb z migrace. U ostatních tabulek tato funkce nedává smysl, protože se jedná o provozní tabulky bez detailních informací, které by bylo potřeba ukládat do externích souborů.

Exportovat je možné ve formátu CSV, případně do schránky uživatele. Druhý export je následně možné vložit do aplikace Excel pro případné uložení do souboru. Při exportu jsou vybírána data podle nastavené filtrace, ale na rozdíl od jejich zobrazení se udělá plnohodnotný výběr přes všechna data, ne jen zmenšený výřez toho co uživatel vidí.

7.5 Dokončení implementace

Implementací posledních stránek aplikace byla dokončena většina funkcí naplánovaných ve vývojové analýze, která proběhla před vytvořením aplikace. Aplikace aktuálně umožňuje následující operace:

- Možnost zobrazit aktuální dění migrací na hlavní stránce aplikace.
- Možnost vytvořit a editovat migrační profil, zobrazit informace o jeho kvalitě a napovědět v případě chybějícího nastavení.
- Možnost schovat, nebo zobrazit pomocná okna při editaci migračního profilu. Možnost změnit pozici výstupních informací z editoru podle potřeby uživatele. Toto nastavení se zapisuje do úložiště prohlížeče, tudíž zůstává trvale uložené.
- Možnost podle potřeby přeskakovat v částech migračního profilu při jeho editaci podle názvů elementů a jejich pozice.
- Možnost vytvořit a editovat migrační úlohu, ručně jí spustit, nebo naplánovat její automatické spuštění. Při plánování úlohy je uživateli zobrazen přeložený cron, kterým se plánuje spuštění do lidsky čitelného jazyka.
- Možnost importovat podpůrné migrační slovníky, které se používají při migraci. Tyto slovníky je možné nahrát v tabulkovém tvaru CSV, dále je možné je ručně vytvořit, editovat, mazat, nebo z aplikace vyexportovat znovu ve formátu CSV.
- Možnost zobrazit záznamy událostí, které nastaly při migraci dat, umožnit jejich export ve formátu CSV nebo do schránky. Data je možné filtrovat podle potřeby, například podle identifikátoru úlohy.

7.6 Problémy spojené s použitím technologie Blazor

Blazor je stále nová technologie, tudíž obsahuje některé problémy, které se při vytváření aplikace musí řešit. V této sekci autor zmiňuje problémy, které nastaly při implementaci Blazor aplikace ve variantě WebAssembly. Všechny problémy souvisely s použitým IDE. Pro vytvoření aplikace byl použit nástroj Visual Studio 2022 verze 17.1.0 ve variantě Community Edition. Jedná se o problémy:

1. Problém interpretovat větší a složitější Blazor zdrojové kódy. Pokud se na problém narazí, tak se špatně či vůbec nezobrazuje zvýraznění syntaxe a kontextové napovídání v IDE.

2. V případě chyb na stránkách Blazor, které nejsou otevřené v editoru, nástroj automaticky nezobrazí informace o problému. To ani v případě chybné kompilace. To znamená, že pokud selhává sestavení aplikace z neznámého důvodu, tak je potřeba projít a zkontrolovat všechny .razor soubory.

3. V případě přesunutí, nebo přejmenování třídy pomocí funkce pro refaktorování, může dojít k rozbití stránek Blazor. Problém nastává, pokud se stránky odkazují na nějakou ze změněných tříd. Problém se projevuje tím způsobem, že se mezi deklaraci `using` a konkrétní `namespace` vloží několik tabulátorů, které danou stránku udělají nevalidní.

4. Velkou část změn na stránkách Blazor není možné pomocí funkce *Hot Reload* propsat do běžící aplikace, ale je potřeba udělat restart.

5. Nakonec, v případě použití Blazor WebAssembly, nejsou podporované ladící (debug) funkce bez úprav a to pouze v podporovaných prohlížečích založených na technologii chromium. V případě Blazor Server problém neexistuje. Spíše než o problém se jedná o vlastnost toho, že aplikace běží v prohlížeči uživatele, ale stejně se jedná o situaci, která může programátora nepříjemně překvapit.

Kapitola 8
Zajímavá řešení

Tato kapitola slouží především jako inspirace a ukázka částí kódů, které nebylo možné ukázat v jiných částech bakalářské práce. Konkrétně jsou popsány tyto zajímavosti:

Rozšíření pro vyhazování výjimek Implementace validace argumentů ve veřejných metodách za použití *extensions* pro vyhazování výjimek.

Middleware pro zachytávání výjimek Implementace *middleware* pro webové rozhraní, které odstraňuje potřebu *try–catch* bloků a tím zvyšuje čitelnost v kódu kontroleru.

8.1 Rozšíření pro vyhazování výjimek

Od verze .NET 6 je možné vytvářet metody se znalostí názvů výrazů, které byly předány jako argumenty do těchto metod. To je možné pomocí nového atributu `CallerArgumentExpression`. Této vlastnosti bylo využito při implementaci validací parametrů veřejných metod. K implementaci byly použity *extensions*, které C# standardně podporuje viz útržek kódu 8.1. Použití rozšíření je možné vidět na útržku kódu 8.2.

Výpis kódu 8.1 Rošíření ThrowIfNull

```
public static void ThrowIfNull<TArgument>(
    this TArgument argument,
    [CallerArgumentExpression("argument")] string paramName = null)
{
    if (argument is null)
    {
        throw new ArgumentNullException(paramName);
    }
}
```

Výpis kódu 8.2 Využití rozšíření ThrowIfNull

```
public async Task<TType> Get(TIdentifier identifier)
{
    identifier.ThrowIfNull();
    ...
}
```

Výhodou tohoto přístupu je jednoduchost použití a odstranění duplicit pro kontroly. U složitějších kontrol, kterou je například metoda `ThrowIfNotValid<TArgument>` pro validaci objektu, se ušetří několik jednotek řádků kódu pro každý ověřovaný parametr.

8.2 Middleware pro zachytávání výjimek

Definice z dokumentace „Middleware je software, který je vložen do aplikační pipeline a zpracovává požadavky a odpovědi.“ [23]. V případě webové aplikace byl *middleware* použitý pro zjednodušení logiky v metodách implementovaných kontrolery. Konkrétně za účelem přesunutí *try–catch* bloku mimo kontroler do připraveného *middleware*. Porovnání je možné vidět na útržcích kódu před 8.3 a po 8.4.

Výpis kódu 8.3 Kontroler před implementací middleware

```
public async Task Update(string jobId, [FromBody] JobDto job)
{
    try
    {
        job.Id = jobId;
        job.ThrowIfNotValid();
        await _jobService.UpdateJobAsync(job);
    }
    catch (ArgumentException ex)
    {
        ...
    }
    catch (NotFoundExcetion ex)
    {
        ...
    }
    catch
    {
        ...
    }
}
```

Výpis kódu 8.4 Kontroler po implementaci middleware

```
public async Task Update(string jobId, [FromBody] JobDto job)
{
    job.Id = jobId;
    job.ThrowIfNotValid();
    await _jobService.UpdateJobAsync(job);
}
```

Použitím *middleware* je možné definovat chování pro všechny očekávané výjimky a odpovědi ze strany serveru na klienta. Při vytváření *middleware* je potřeba implementovat veřejnou asynchronní metodu `Task InvokeAsync(HttpContext httpContext)`, která se v *pipeline* dotazu vykonává. Její výsledek je použitý při volání dalších *middleware*. Pro zajištění očekávaného tvaru tříd bylo vytvořeno rozhraní `IMiddleware` s definicí této metody. Příkladem implementace *middleware* je třída `ExceptionsHandlerMiddleware` viz útržek kódu 8.5. Implementace útržku byla inspirována podobným řešením z webu Code Maze [24]. Z útržku je vidět, že při odchytávání výjimek se očekávají chyby, které byly vytvořeny pro potřeby webového rozhraní. Tyto chyby pak určují odpověď na klienta. Velká výhoda tohoto přístupu je definice veškerého chování na jednom místě pro všechny kontrolery.

Výpis kódu 8.5 Implementace třídy ExceptionsHandlerMiddleware

```
public class ExceptionsHandlerMiddleware : IMiddleware
{
    private readonly RequestDelegate _requestDelegate;
    private readonly ILogger _logger;

    public ExceptionsHandlerMiddleware(RequestDelegate requestDelegate,
        ILogger<ExceptionsHandlerMiddleware> logger)
    {
        _requestDelegate = requestDelegate;
        _logger = logger;
    }

    public async Task InvokeAsync(HttpContext httpContext)
    {
        try
        {
            await _requestDelegate(httpContext);
        }
        catch (NotFoundException ex)
        {
            _logger.LogInformation($"Nebyl nalezen zdroj: {ex}");
            await WriteResponseAsync(httpContext, ex,
                (int)HttpStatusCode.NotFound);
        }
        catch (ServerException ex)
        {
            _logger.LogError(ex, $"Nastal ocekavany problem: {ex}");
            await WriteResponseAsync(httpContext, ex,
                (int)HttpStatusCode.BadRequest);
        }
        catch (AuthorizationException ex)
        {
            _logger.LogError(ex, $"Nastala chyba autorizace: {ex}");
            await WriteResponseAsync(httpContext, ex,
                (int)HttpStatusCode.Unauthorized);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"Nastal neocekavany problem: {ex}");
            await WriteResponseAsync(httpContext, ex,
                (int)HttpStatusCode.InternalServerError);
        }
    }

    private static async Task WriteResponseAsync(HttpContext context,
        Exception exception, int statusCode)
    {
        var details = new ErrorDetail
        {
            StatusCode = statusCode,
            Message = exception.Message
        };

        context.Response.ContentType = "application/json";
        context.Response.StatusCode = statusCode;

        await context.Response.WriteAsJsonAsync(details);
    }
}
```

Kapitola 9
Testování a dokumentace

9.1 Testování

Při vývoji je proces testování jeden ze základních kamenů úspěšného projektu. Testování je možné rozdělit do dvou skupin a to manuální a automatické. V případě manuálního testování se může jednat o manuální procházení uživatelského rozhraní daného scénářem či případy užití, a postupné ověření implementovaných funkcionalit. Nevýhodou tohoto přístupu je rychlost a cena. Zapojením člověka můžeme získat určité výhody, které automatizované testování nemá. Tím je lidský názor na funkčnost aplikace a schopnost vidět problémy, které stroj nevidí. Určitě je vhodné, alespoň v jedné fázi vývoje, takový test provést. Z tohoto důvodu byla aplikace důkladně otestována podle všech případů užití viz příloha A. To znamená, že všechny případy užití byly postupně otestovány na aplikaci v pořadí, které dávalo smysl. A tím bylo ověřeno, že aplikace splňuje slíbené funkční požadavky.

Druhým případem je automatické testování. V případě vyvíjené aplikace byly zapojeny následující automatizované způsoby testování:

Jednotkové testování Především známé jako *Unit testing*. Jedná se o testy na malých částech zdrojového kódu, u kterých má programátor plnou kontrolu nad vstupními daty. Pro oddělení závislosti na další implementaci, kterou může být například získávání dat z databáze, se používají *mock* nástroje. Tyto nástroje jsou schopné zajistit přesně očekávané chování od závislých částí kódu.

Integrační testování Jedná se o testování propojených služeb a částí kódu. Na rozdíl od *Unit testing* se zde kontroluje propojení databáze a jakým způsobem jsou data získávána. Zajímavostí je, že *mock* nástroje se při integračním testování používají také. Například při testování celého webového rozhraní viz ukázka kódu 9.1.

Za zmínku stojí i statická analýza kódu. Jedná se o proces kontroly zdrojového kódu bez jeho kompilace, který se snaží odhalit rozpory ve stylu programování, chyby ve zdrojovém kódu a též problémy nazvané *code smell* (pach v kódu). *Code smell* jsou myšleny charakteristiky kódu, které mohou značit větší problémy v logice, nebo programování. Analýzu mohou provádět některé IDE automaticky, nebo po instalaci rozšíření, jako je například ReSharper pro Visual Studio. Dále existují NuGet balíčky, které je možné nainstalovat do projektu stejně jako jiné knihovny pomocí NuGet manažeru, kterým se říká analyzátory. Nakonec se může jednat o samostatné aplikace, které statickou analýzu provádí automaticky. Tyto aplikace je možné zapojit do vývoje ve fázi nahrávání zdrojového kódu do sdíleného repozitáře. V případě vyvíjené aplikace byl použit standardní analyzátor, který je použit s každým novým projektem od .NET 5, konkrétně se jedná o `Microsoft.CodeAnalysis.NetAnalyzers`.

9.1.1 Automatické testování aplikace

Při vývoji byly použity dvě techniky automatického testování: *unit testing* a integrační testy. V obou případech byl použit stejný testovací *framework* MSTest v2. Ten je na rozdíl od svého předchůdce distribuován jako NuGet balíček. Problém předchozí verze spočíval v tom, že bylo potřeba mít nainstalované vývojové nástroje distribuované s aplikací Visual Studio, takže při použití jiného IDE bylo potřeba nainstalovat i Visual Studio.

MSTest provádí testování na veřejných třídách označených atributem `TestClass`. Testují se veřejné metody označené atributem `TestMethod`. Testy byly při vytváření rozděleny na tři části: příprava, provedení a test. V první části se připravují potřebná data, ve druhé části je vykonávána testovaná funkcionalita a v poslední části je provedena kontrola výsledků viz útržek kódu 9.2.

Při integračním testování byly ověřeny kontrakty mezi webovým rozhraním a službami na straně webové aplikace. K tomu byla použitá třída `WebApplicationFactory`. Třída je součástí NuGet balíčku `Microsoft.AspNetCore.Mvc.Testing`. `WebApplicationFactory` umožňuje vytvořit webové rozhraní jako instanci HTTP klienta, se kterým pracují vytvořené služby. Při vytváření klienta je možné modifikovat nastavení služeb, které klasicky probíhá při startu aplikace ve třídě `Program` viz útržek kódu 9.1. Na útržku je vidět konfigurace továrny pro webové rozhraní, která se používá při vytvoření testovacího webové klienta. Probíhá zde nahrazení konfigurace databáze za variantu „in memory“. Metoda `RemoveDbContextFactory` je vlastní rozšíření, které najde konkrétní `DbContextFactory` a odstraní ho z nastavení.

Výpis kódu 9.1 Konfigurace třídy WebApplicationFactory

```
_factory = new WebApplicationFactory<server.Program>()
    .WithWebHostBuilder(builder =>
    {
        builder.ConfigureServices(services =>
        {
            services.RemoveDbContextFactory<AppDbContext>();
            services.AddDbContextFactory<AppDbContext>(opt =>
                opt.UseInMemoryDatabase(DatabaseName));
        });
    });
```

Z útržku kódu 9.2 je možné vidět jednoduchý integrační test.

Výpis kódu 9.2 Integrační test třídy ProfileService a webového rozhraní

```
[TestClass]
public class ProfileContract_Tests
{
    [TestMethod]
    public async Task ShouldReturnDefaultProfile()
    {
        // Priprava
        var client = _factory.CreateClient();
        var service = new client.ProfileService(
            new EmptyLogger<client.ProfileService>(), client);

        // Provedeni
        var profile = await service.GetDefault();

        // Test
        Assert.IsNotNull(profile);
        Assert.IsTrue(profile.Data.Length > 0);
    }
}
```

9.2 Dokumentace

Práce je dokumentována dvěma způsoby. Prvním, hlavním způsobem, je technická dokumentace. Ta je vygenerována ze zdrojových kódů aplikace pomocí nástroje Doxygen. Doxygen umožňuje generovat dokumentaci ze zdrojových kódů nejrůznějších jazyků od C po Python. Veškeré generování je řízeno nastavením, kterému se říká Doxyfile. Doporučeným způsobem, jakým připravit toto nastavení, je použít aplikaci Doxywizard, která se instaluje společně s nástrojem Doxygen. Doxywizard usnadňuje natavení generování tím, že s uživatelem postupně projde důležité možnosti a umožní jednoduše vybrat co je potřeba. Doxyfile aplikace je uložený vedle technické dokumentace na přiloženém médiu. V případě migrační aplikace je dokumentace generována ve tvaru HTML stránek. Před generováním je potřeba mít nainstalovaný balíček GraphViz, který Doxygen používá ke generování diagramů částí z aplikace.

Best practice při vývoji je psaní komentářů ke zdrojovému kódu. V případě migrační aplikace byly komentáře napsány nad všechny veřejné třídy, rozhraní, metody, položky a členy. Pokud to bylo potřeba, tak byl popsán i nepřehledný či složitější privátní zdrojový kód. Slovy veřejný a privátní jsou myšleny modifikátory přístupu. Pod veřejným se schovává `public` a pod privátním všechno ostatní. Při psaní byly dodržovány standardy viz dokumentace [25]. Psaní komentářů je klíčové pro kvalitní technickou dokumentaci, protože slouží jako hlavní zdroj informací při generování obsahu dokumentace. Doxygen navíc podporuje stejné *tags* (štítky), které jsou vysvětlené ve zmíněné dokumentaci.

Druhým způsobem dokumentace je instalační příručka, která je umístěna v přílohách bakalářské práce. Viz příloha D. Instalační příručka popisuje způsoby instalace a provozu webové aplikace podle případného použití aplikace v praxi. Konkrétně se jedná o instalaci aplikace na platformě Windows ve formě Microsoft Služby, která běží automaticky, nebo jako konzolové aplikace, kterou spouští uživatel ručně. Všechny projekty migrace v oddělení digitalizace, které byly do teď provedeny, proběhly pouze na této platformě.

Kapitola 10
Závěr

S ohledem na cíle bakalářské práce, které byly zmíněny v úvodní kapitole, byly vypracovány všechny kroky k naplnění hlavního cíle bakalářské práce. Byla popsána doména spisových služeb. Dále byl definován a popsán proces migrace v kontextu spisových služeb. Proběhla analýza procesu migrace, ze kterého vzešly požadavky na migrační aplikaci. Jmenovitě: editor migračního profilu, editor a správce migračních slovníků, správa migračních úloh, možnost manuálního i automatizovaného spouštění migračních úloh, funkce pro vyhledání v chybových záznamech migrace s možností exportu a funkce pro přehled nad aktuálně běžícími migracemi se statistikou. Byl proveden návrh aplikace, který se skládal z návrhu architektury, popsání vybrané technologie, pokrytí požadavků pomocí případů užití a vytvoření obrazovek pro pokrytí požadavků a případů užití. Poté byly implementovány všechny funkční požadavky včetně propojení s existující migrační knihovnou a dodrženy technické požadavky. Aplikace byla při vývoji testována pomocí jednotkového a integračního testování. Při vývoji také probíhala nad zdrojovými kódy pravidelně statická analýza kódu. Dále byla aplikace otestována pomocí funkčních testů proti funkčním požadavkům z analýzy. Tím bylo zaručeno, že jsou všechny funkční požadavky implementovány. Také byla vytvořena technická dokumentace z plně komentovaných zdrojových kódů aplikace. Nakonec byla vytvořena instalační příručka pro očekávané způsoby použití aplikace.

Při implementaci byly odhaleny další zajímavé funkcionality, které by měli být v budoucnu implementovány. První je funkce pro spuštění testovacího běhu migrace z editoru migračního profilu s možností předat data pro migraci přímo z editoru. Tato funkce by ještě více urychlila přípravu migračního profilu, protože by bylo možné testovat profil postupně při jeho tvorbě. Další funkcí je přidání upozornění pro změny stavů migračních úloh i mimo hlavní stránku nebo seznam úloh. To by přineslo větší přehled do aktuálního stavu běžících migrací, protože by uživatel nemusel neustále aktualizovat stránky aplikace a čekat na výsledek.

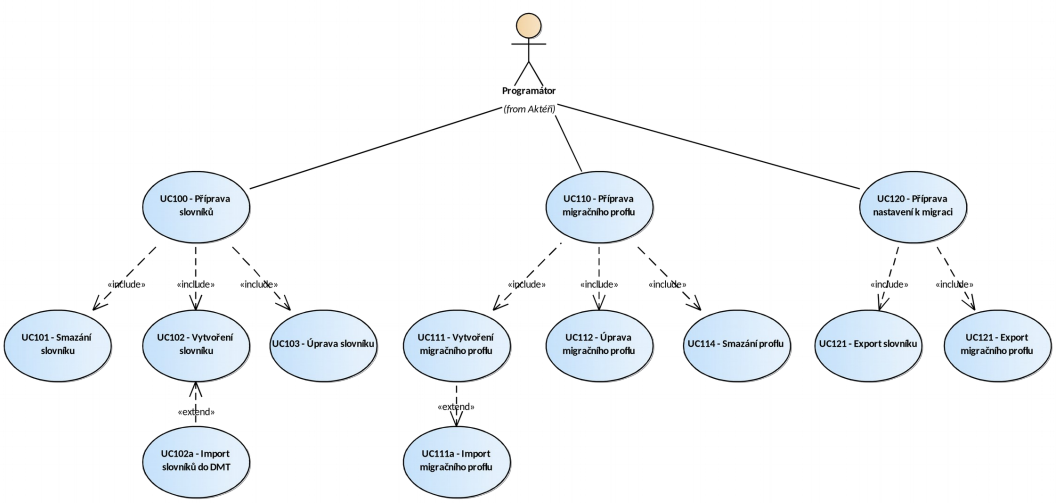
To znamená, že byly naplněny všechny požadavky ze zadání bakalářské práce. Výsledná aplikace usnadňuje migraci všemi implementovanými funkcemi, které byly zmíněny. K vytvoření byla použitá nová Microsoft technologie Blazor WebAssembly a ASP.NET Core, oboje v nejnovější verzi .NET. Tudíž byla dodržena slíbená technologie. Původní knihovna byla aktualizována na novější verzi .NET a upravena pro potřeby migrační aplikace. Aplikace byla otestována a dokumentována. V tuto chvíli je možné aplikaci použít pro projekty migrace v oddělení digitalizace firmy ICZ.

Příloha A

Případy užití

A.1 Programátor (fáze přípravy)

Obrázek A.1 Programátor – Fáze přípravy



UC100 – Příprava slovníků

Přípravou je myšlen proces vytvoření a editace podpůrných migračních slovníků.

UC101 – Smazání slovníku

Předpoklad: Existuje slovník, který je potřeba vymazat.

1. Uživatel otevře správu slovníků a vybere slovník.
2. Uživatel klikne na volbu pro smazání slovníku a potvrdí akci.

3. Pokud se jednalo o chybu, tak uživatel může přerušit mazání slovníku.
4. Pokud uživatel potvrdí smazání, tak aplikace smaže slovník.

UC102 – Vytvoření slovníku

Předpoklad: Je potřeba vytvořit nový slovník ručně v aplikaci.

1. Uživatel otevře správu slovníků a vybere možnost pro přidání nového slovníku.
2. Aplikace otevře obrazovku pro vytvoření slovníku.
3. Uživatel pomocí tlačítka na přidání hodnoty vyvolá dialog, ve kterém může přidat položky do hodnotu slovníku. Po dokončení uživatel uloží hodnotu.
4. Po dokončení slovníku uživatel klikne na tlačítko uložit slovník.
5. Uživatel provede výběr jména a po potvrzení se slovník uloží.
6. Aplikace kontroluje, zda jsou názvu slovníků unikátní a nedovolí přidat slovník s neunikátním názvem.

UC102a – Import slovníků do DMT

Předpoklad: Uživatel má slovníky připravené v souboru ve vhodném formátu pro import.

1. Uživatel otevře správu slovníků a vybere a vybere možnost importovat slovník.
2. Uživatel z disku vybere slovník pro import.
3. Aplikace načte vybraný slovník a zobrazí uživateli odhadovaný tvar po importu.
4. Uživatel podle potřeby změní oddělovače slovníku, kódování jazyka a jiné nastavení.
5. V případě slovníku s existujícím názvem aplikace upozorní uživatele a neumožní uložení bez opravy.
6. Uživatel potvrdí nahrání slovníků a aplikace tak učiní.
7. V případě nějaké chyby importu se uživateli zobrazí srozumitelná informace.

UC103 – Úprava slovníku

Předpoklad: Uživatel zjistil novou informaci, která mu odhalila potřebu změnit nastavení slovníku.

1. Uživatel otevře správu slovníků a vybere slovník, který je potřeba změnit.
2. Uživatel přidá nové položky, nebo změní existující položky ve slovníku.
3. Po dokončení všech úprav uživatel provede uložení slovníku.

UC110 – Příprava migračního profilu

Profil je připravován podle analýzy zdrojových dat a podle znalosti cílového systému.

UC111 – Vytvoření migračního profilu

Předpoklad: Migrační profil jako takový neexistuje a je potřeba ho vytvořit.

1. Uživatel otevře správu migračních profilů a klikne na tlačítko pro přidání profilu.
2. Uživatel zadá název nového profilu a potvrdí výběr.
3. Aplikace ověří správnost a unikátnost vybraného názvu. Pokud dojde k nějaké chybě, tak uživatele upozorní a nedovolí mu profil přidat.
4. Pokud je vše v pořádku, tak se profil uloží do aplikace.
5. Aplikace otevře nový profil v editoru migračního profilu.

UC111a – Import migračního profilu

Předpoklad: Uživatel má k dispozici existující migrační profil, který je možné přidat do aplikace.

1. Uživatel otevře správu migračních profilů a klikne na tlačítko importu profilu.
2. Aplikace umožní uživateli nahrát vybraný profil pomocí souborového dialogu.
3. Aplikace předvyplní název profilu podle názvu předaného souboru.
4. Uživatel v případě potřeby změní název profilu a potvrdí import.
5. Aplikace ověří, že je název profilu unikátní. Pokud není, tak uživatele upozorní a dotáže ho na nový název.
6. Po potvrzení je profil uložen do aplikace.

UC112 – Úprava migračního profilu

Předpoklad: V nástroji již existuje profil, který je potřeba upravit.

1. Uživatel otevře správu migračních profilů.
2. Uživatel vybere migrační profil a klikne na tlačítko úpravy profilu.
3. Aplikace profil otevře v editoru migračního profilu.
4. Uživatel provede úpravy podle potřeby.
5. Pokud je uživatel spokojený se svými změnami, tak pomocí tlačítka uložit požádá o uložení profilu.
6. Pokud je vše v pořádku, tak aplikace provede uložení profilu.
7. Pokud si uživatel změny rozmyslel, tak může pomocí tlačítka všechny změny zahodit.

UC114 – Smazání profilu

Předpoklad: Uživatel má důvod vymazat existující profil migrace.

1. Uživatel otevře správu migračních profilů a vybere migrační profil, který chce vymazat.
2. Uživatel klikne na tlačítko vymazání profilu.
3. Aplikace se dotáže uživatele, jestli chce doopravdy vymazat profil.
4. Po potvrzení je profil vymazaný z aplikace.

UC120 – Příprava nastavení k migraci

Jedná se o fázi, ve které je potřeba všechny vytvořené nastavení exportovat z aplikace a uložit.

UC121 – Export migračního profilu

Předpoklad: Existuje profil, který je z nějakého důvodu potřeba exportovat z aplikace.

1. Uživatel otevře správu profilů a vybere profil, který chce exportovat.
2. Uživatel vybere možnost uložit profil.
3. Aplikace vybídne uživatele k zadání místa uložení.
4. Po potvrzení aplikace provede uložení profilu.

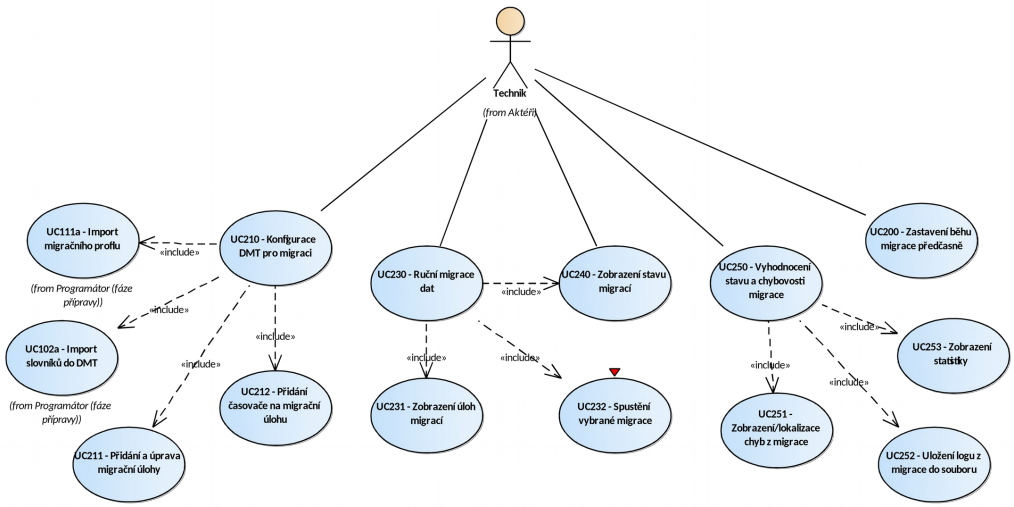
UC122 – Export slovníku

Předpoklad: Existuje slovník, který je potřeba z nějakého důvodu exportovat z aplikace.

- Uživatel otevře správu slovníků a vybere slovník, který chce exportovat.
- Aplikace vygeneruje slovník ve tvaru CSV souboru pojmenovaný podle názvu slovníku.
- Aplikace vybídne uživatele k zadání místa uložení.
- Po potvrzení aplikace provede uložení slovníku.

A.2 Technik

Obrázek A.2 Technik



UC200 – Zastavení běhu migrace předčasně

Předpoklad: Uživatel potřebuje z nějakého důvodu zastavit běžící migraci.

1. Uživatel zobrazí seznam běžících úloh a vybere úlohu, kterou je potřeba zastavit.
2. Uživatel pomocí tlačítka pro zastavení úlohy požádá aplikaci o její zastavení.
3. Aplikace přijme požadavek a provede zastavení běžící úlohy. Zastavení úlohy se provede až po dokončení migrace aktuálně rozpracovaného objektu.
4. Aplikace informuje uživatele o zastavení úlohy.

UC210 – Konfigurace DMT pro migraci

Konfigurace v DMT se dá rozdělit na dva typy: interní konfigurace a externí konfigurace. Mezi interní konfiguraci patří přidání migračních profilů, přidání a nastavení úloh migrace a doplnění migračních slovníků. Mezi externí naopak patří nastavení logování, nasměrování na databázi aplikace a základní nastavení migrační knihovny.

UC211 – Přidání a úprava migrační úlohy

Předpoklad: Uživatel ví, kterou úlohu je potřeba přidat pro potřeby migrace. V aplikaci existuje migrační profil.

1. Uživatel otevře správu migračních úloh.
2. Uživatel přidá migrační úlohu a pojmenuje jí unikáním názvem.
3. Uživatel podle potřeby uživatel přidá 1 a více migračních profilů do úlohy.
4. Uživatel nastaví pořadí spouštěných migračních profilů podle potřeby.
5. Uživatel může nastavit automatické spouštění úlohy viz UC212.
6. Uživatel potvrdí uložení úlohy.
7. Aplikace ověří, že je název unikátní a provede další kontroly. Pokud je něco špatně, tak upozorní uživatele na problém a nedovolí mu úlohu uložit.
8. Pokud je všechno správně, tak aplikace uloží úlohu.

UC212 – Přidání časovače na migrační úlohu

Předpoklad: V aplikaci existuje úloh. Uživatel chce přidat její automatické spouštění.

1. Uživatel otevře správu migračních úloh a vybere konkrétní úlohu, kterou si přeje upravit.
2. Uživatel vybere možnost přidat nový *trigger* na vybranou úlohu.
3. Aplikace zobrazí uživateli nastavení pro nově přidávaný *trigger*.
4. Uživatel provede nastavení ve formátu cron. Pro bližší kontrolu je cron přeložen do lidského jazyka.
5. Aplikace ověří, že je vše nastaveno správně. Pokud ne, tak uživatele upozorní a nedovolí mu trigger přidat.

6. Po potvrzení se provede dočasné uložení do paměti aplikace. Trigger není propsán do aplikace okamžitě, ale až po potvrzení změn v úloze.

7. Uživatel může přidat další trigger.

8. Po potvrzení změn, aplikace uloží nové informace

UC230 – Ruční migrace dat

Ruční migrací je myšlena činnost migrace vyvolaná uživatelem. Součástí tohoto případu užití je možnost výběru konkrétní úlohy v dialogu, ruční spuštění úlohy pomocí tlačítka a snadná kontrola stavu migrace.

UC231 – Zobrazení úloh migrací

Předpoklad: V aplikaci existuje alespoň jedna úloha s připraveným migračním profilem.

1. Uživatel otevře pomocí volby úloh všechny existující úlohy v aplikaci.

2. Aplikace poskytne uživateli seznam všech existujících úloh včetně jejich stavu a možnosti přejít do nastavení úlohy, profilu, nebo logů úlohy.

UC232 – Spuštění vybrané migrace

Předpoklad: Uživatel si vybral nějakou úlohu v seznamu existujících úloh, vybraná úloha není spuštěná, ale je povolená.

1. Uživatel klikne na tlačítko „spustit úlohu“.

2. Pokud není úloha ve stavu, kdy je možné jí spustit, tak je tlačítko schované

3. Pokud se nepovedlo spustit úlohu, tak je uživatel informován.

4. Aplikace provede okamžité spuštění úlohy a aktualizaci stavů.

UC240 – Zobrazení stavu migrací

Předpoklad: Je spuštěna minimálně jedna migrace.

1. Uživatel otevře aplikaci na hlavní stránce, též zvaný *dashboard*.

2. V *dashboard* je vidět aktuální stav běžících migraci.

3. Uživatel může, pomocí odkazu na názvu úlohy, přejít do nastavení úloh.

UC250 – Vyhodnocení stavu a chybovosti migrace

Důvodem vyhodnocení je zjištění dalších kroků migrace. V případě větší chybovosti bude potřeba provést opravu migračního profilu, nebo jiného nastavení než se bude dále pokračovat. Dalším důvodem je potřeba vědět jak daleko migrace pokročila a jestli se zdárně blíží konci.

UC251 – Zobrazení/lokalizace chyb z migrace

Předpoklad: Byla spuštěna nějaká migrace, při níž se vyskytly chyby. V tomto případě jsou myšleny chyby: špatná transformace, nebo validace dat, chybějící hodnoty ve slovnících, nebo nepodchycený *timeout* při volání API, nebo databáze.

1. Uživatel vybere možnost zobrazení záznamů z migrace.
2. Uživatel pomocí filtru zmenší hledané záznamy z migrace. V případě chyb se jedná o záznamy, které jsou označené jako úroveň WARN a vyšší. Dále je možné vybrat datum chyby, stav migrovaného objektu, název profilu a jiné.
3. Aplikace podle vybrané filtru zobrazí uživateli data rozdělené na stránky.
4. Aplikace dále umožní uživateli nastavit si velikost stránky zobrazených dat.

UC252 – Uložení logu z migrace do souboru

Předpoklad: Uživatel zobrazil zajímavé informace ze záznamu zpracování a chyb. Uživatel chce tyto záznamy uložit.

1. Uživatel má otevřené záznamy z migrace.
2. Uživatel klikne na možnost exportovat.
3. Uživatel vybere možnost uložení do souboru, nebo do schránky.
4. V případě uložení do souboru aplikace nabídne uživateli možnost vybrat si místo uložení.
5. Pokud se při vytváření exportu stane nějaká chyba, tak je viditelně zobrazená uživateli.

UC253 – Zobrazení statistiky

Předpoklad: V aplikaci proběhla migrace v posledních 7 dnech. Existují záznamy, které je možné zobrazit.

1. Uživatel otevře hlavní obrazovku (dashboard).
2. Aplikace uživateli zobrazí informace o zpracování dat za posledních 7 dní – počet migrovaných dat, počet chyb, rychlost migrace.
3. Případě aktuálního zpracování bude probíhat automatická aktualizace.

A.3 Programátor (fáze dokončení)

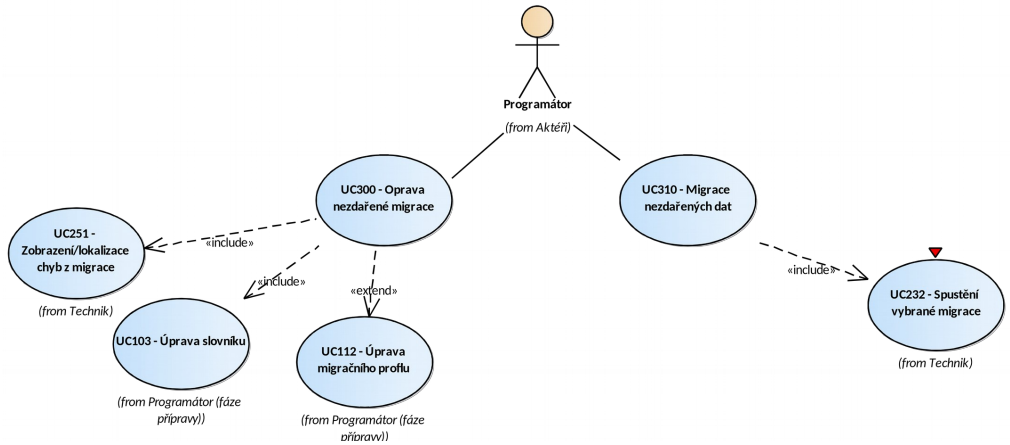
UC300 – Oprava nezdařené migrace

Opravou nezdařených dat jsou myšleny opravy migračního profilu, ruční zásahy do zdrojových dat a aktualizace migračního slovníku.

UC310 – Migrace nezdařených dat

Migrací nezdařených dat jsou myšleny kroky opravy problému: analýza problému, oprava dat, nebo nastavení a znovu spuštění migrace pro vybraná data. Součástí je přesunutí dat k migraci znovu na vstup.

Obrázek A.3 Programátor – Fáze dokončení



A.4 Automat

UC400 – Automatické spuštění migrace

Předpoklad: V aplikaci existuje alespoň jedna aktivní úloha, která má definovaný čas spuštění.

1. Při startu aplikace se aktivní automatické úlohy přidají do plánovače spuštění. Tam vyčkávají na spuštění.
2. Plánovač úloh, též automat, provede ve správný čas spuštění migrační úlohy.
3. Úloha postupně prochází migrační profily, pomocí nichž provádí migraci.
4. Po dokončení migrace se úloha dočasně vypne a čeká na další čas spuštění.

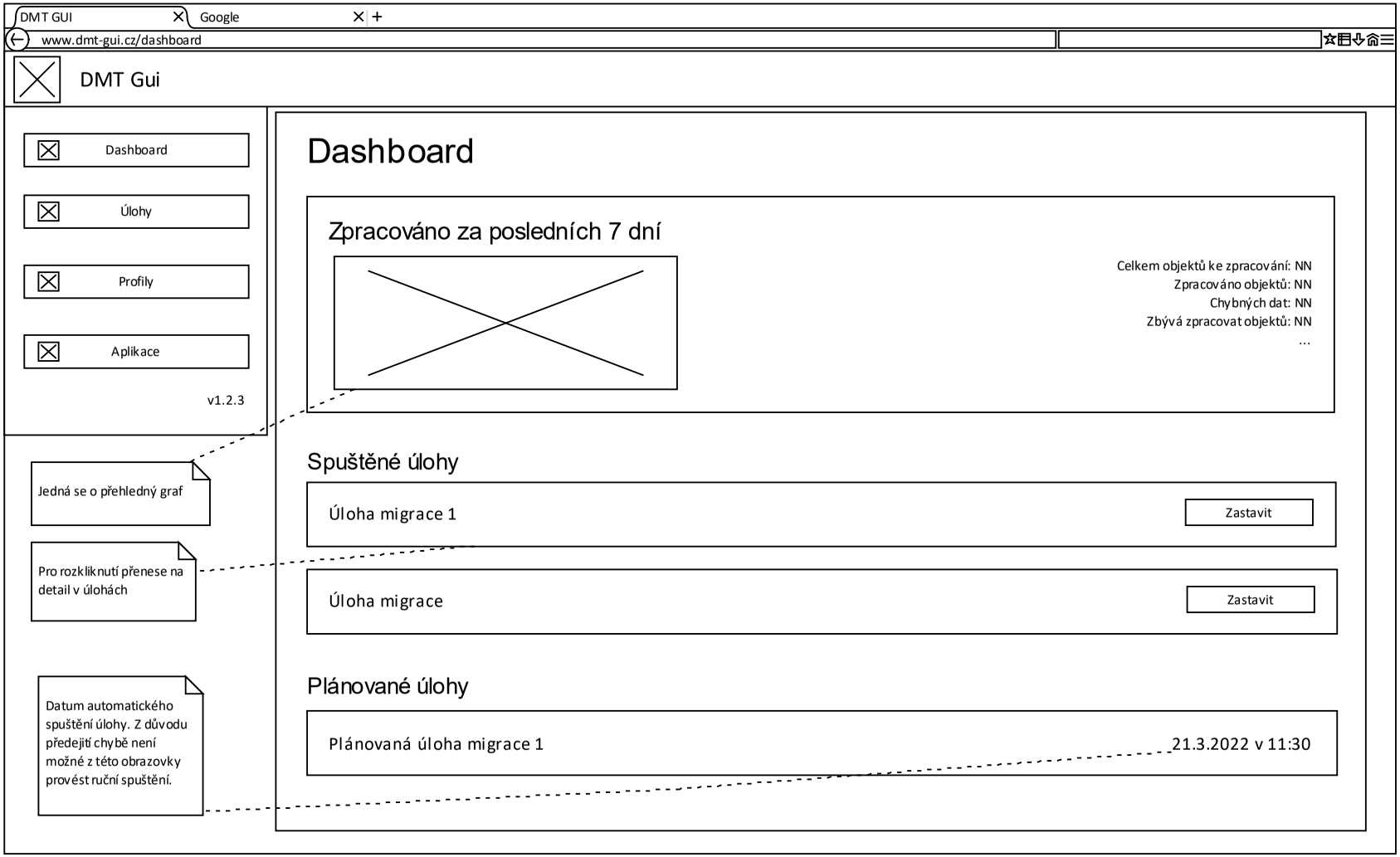
UC410 – Automatické zastavení migrační úlohy

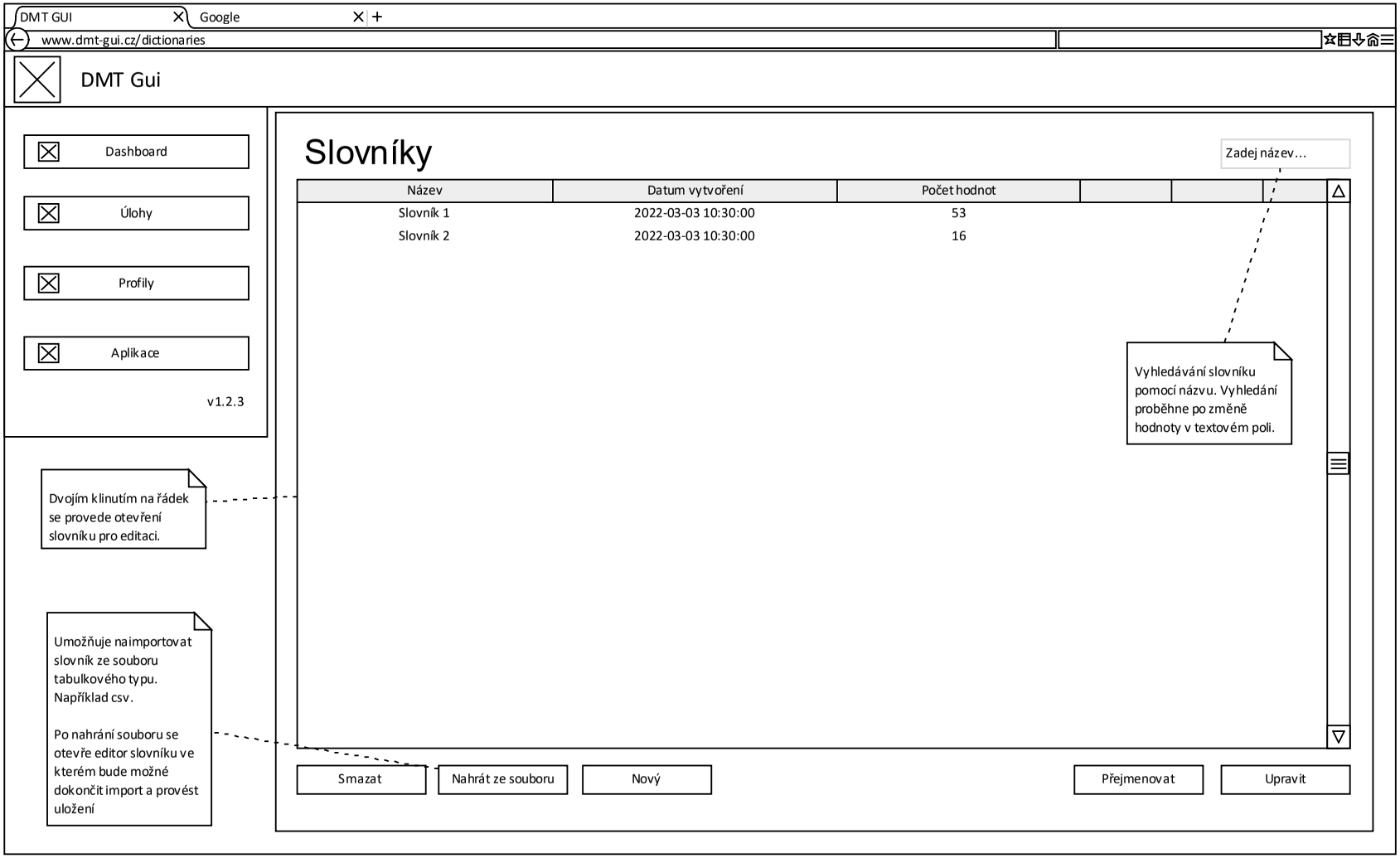
Předpoklad: Pro *trigger* migrační úlohy je definovaná hodnota délky běhu migrační úlohy. Daná úloha běží potřebnou dobu pro její automatické vypnutí.

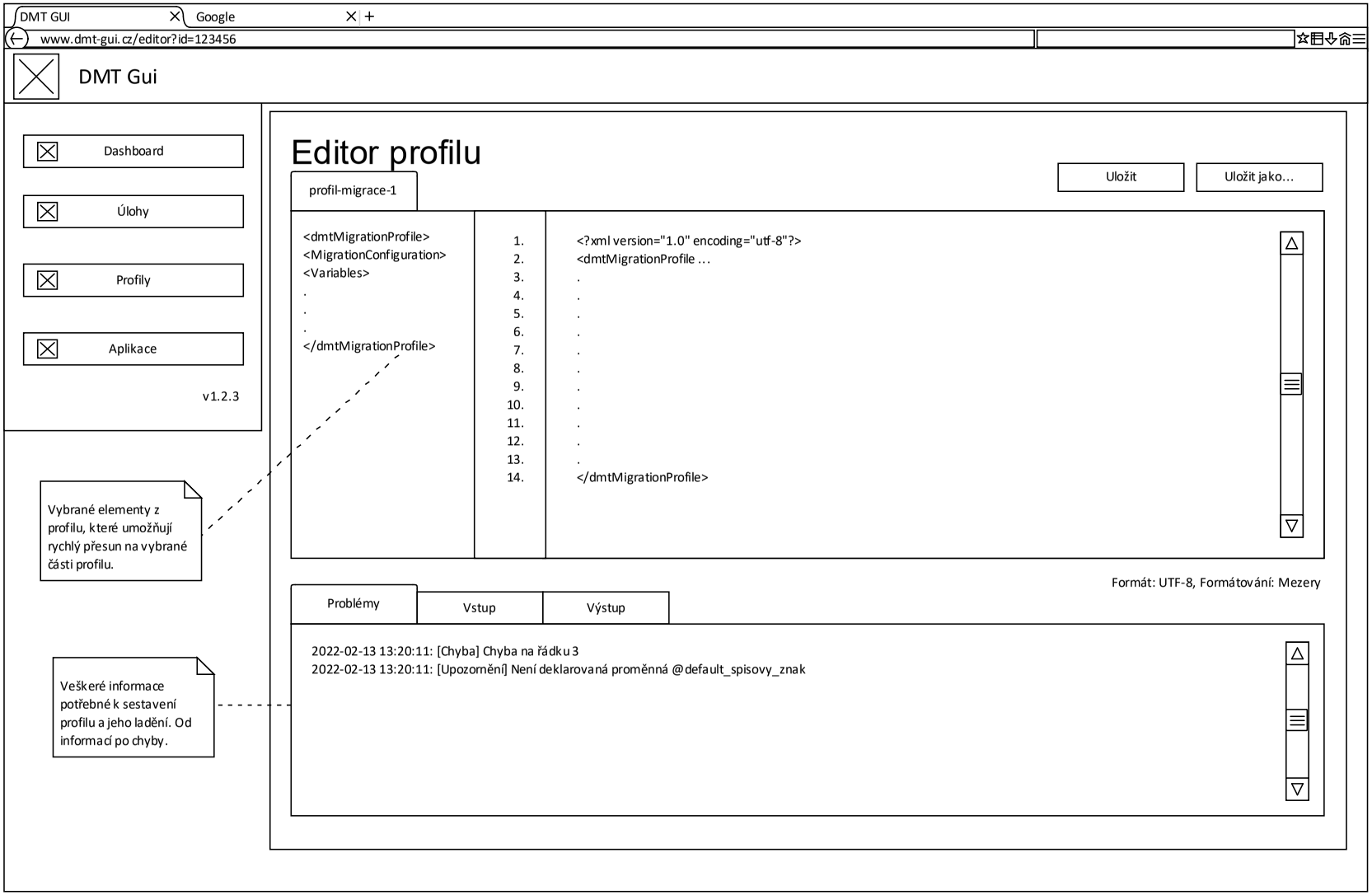
1. Automat zaznamenal, že migrační úloha běží maximální povolenou dobu.
2. Automat zašle požadavek na ukončení úlohy běžící úloze.
3. Migrace se musí vypnout sama z důvodu omezení množství chyb při migraci. Brzké ukončení by mohlo případně ohrozit cílový systém, nebo poškodit migrovaná data.

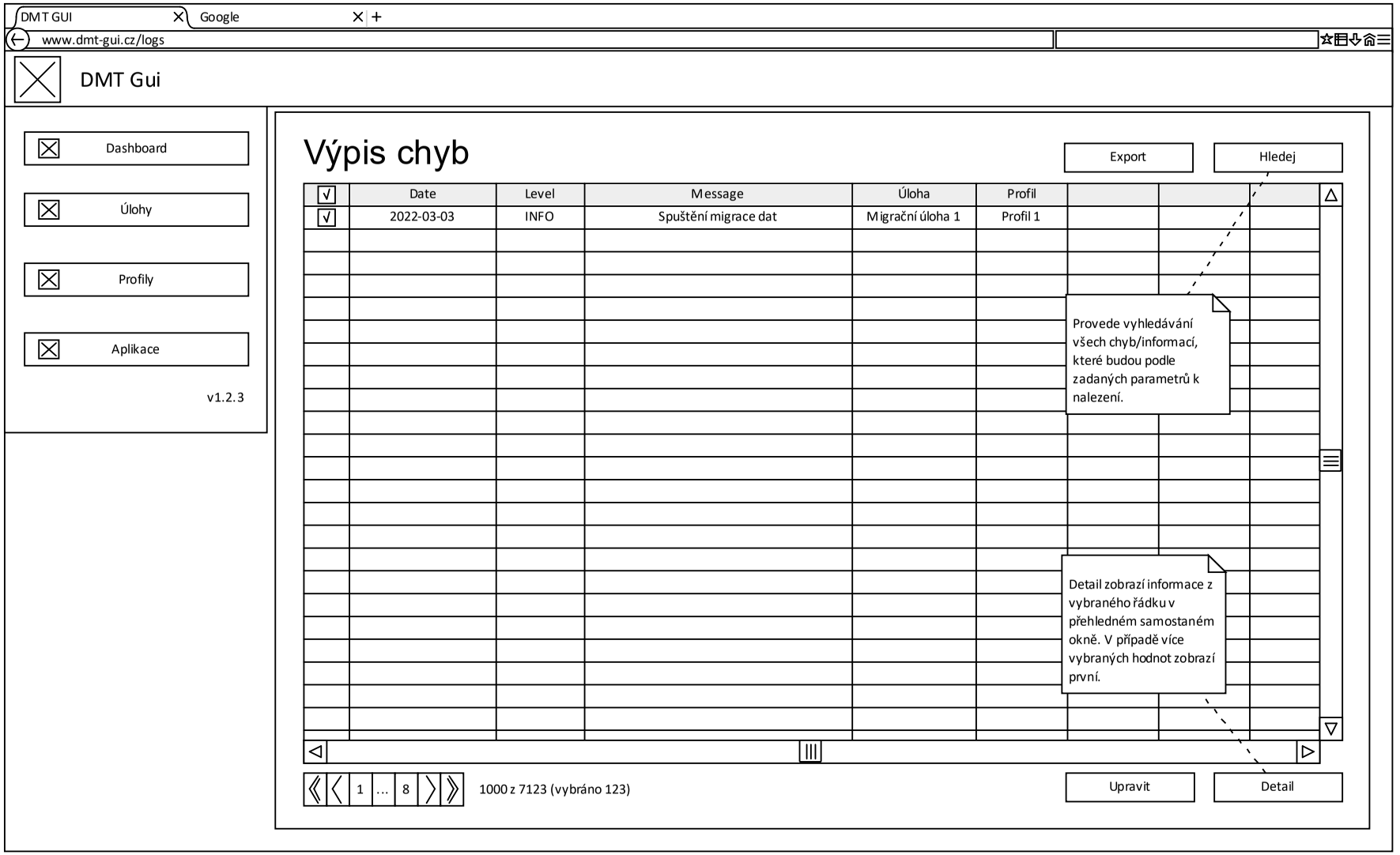
Příloha B

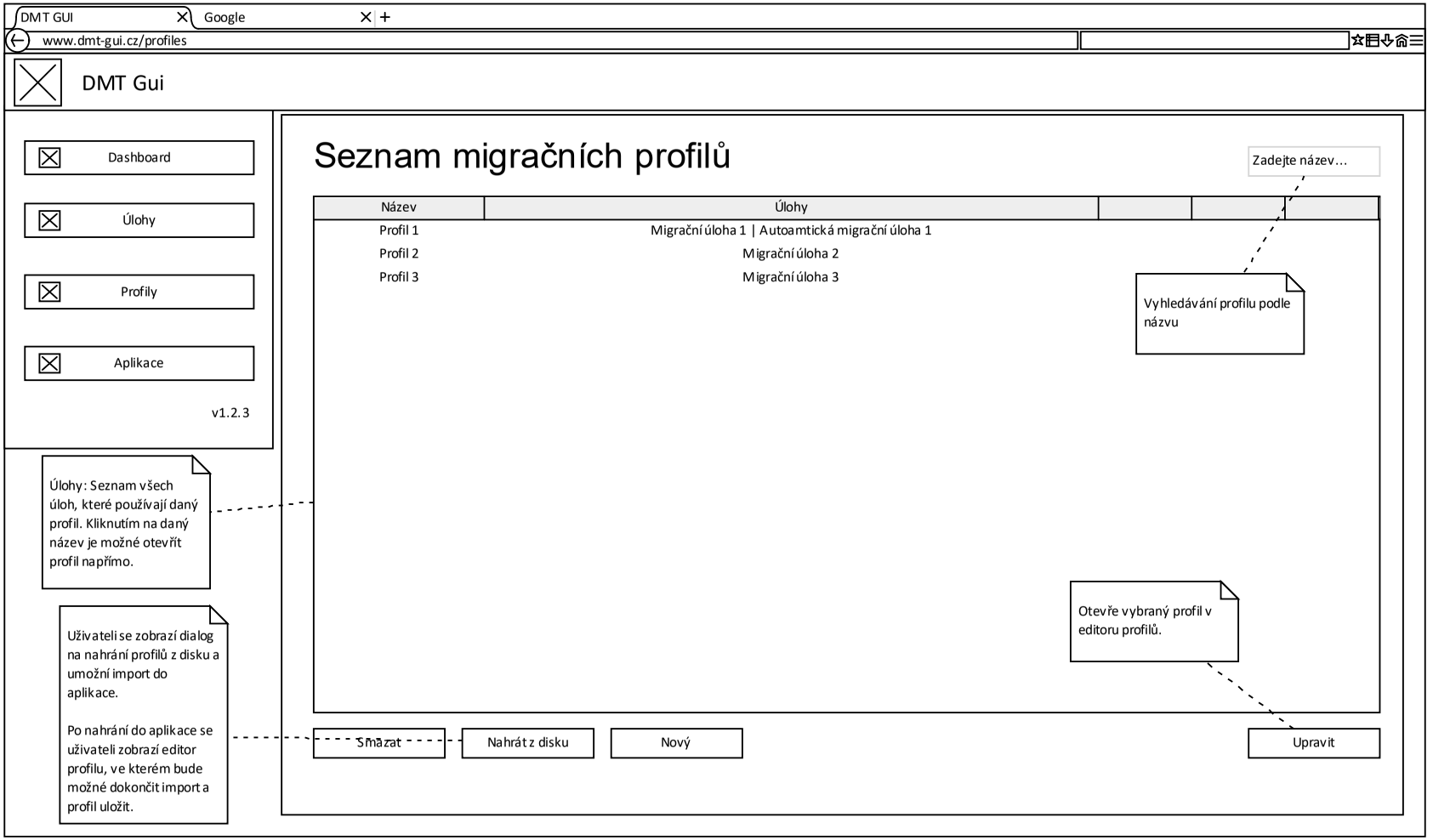
Návrhy obrazovek

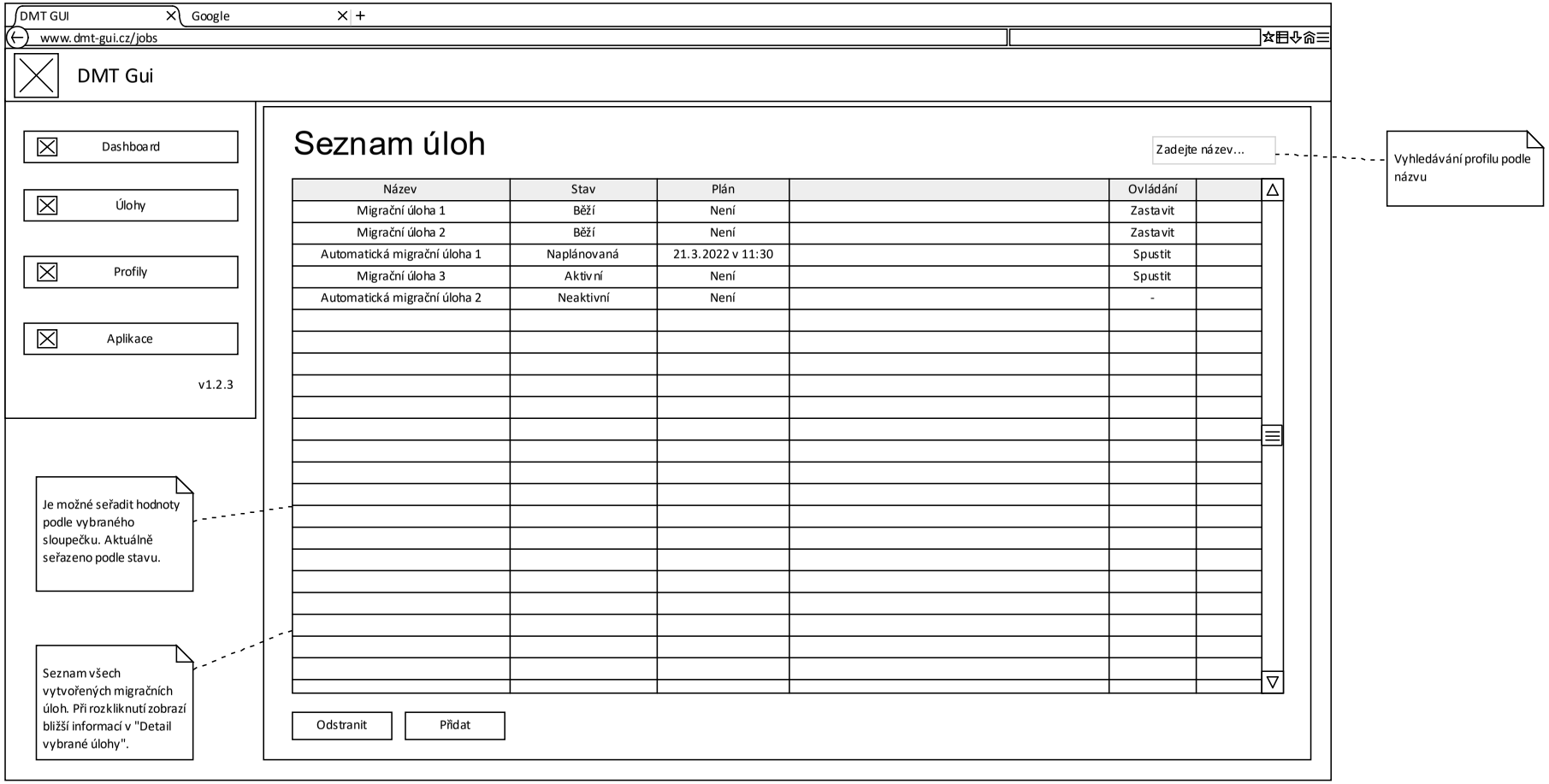

DMT GUI
Google
www.dmt-gui.cz/dashboard
DMT Gui
Dashboard
Úlohy
Profily
Aplikace
v1.2.3
Dashboard
Zpracováno za posledních 7 dní
Celkem objektů ke zpracování: NN
Zpracováno objektů: NN
Chybných dat: NN
Zbývá zpracovat objektů: NN
...
Spuštěné úlohy
Úloha migrace 1
Zastavit
Úloha migrace
Zastavit
Plánované úlohy
Plánovaná úloha migrace 1
21.3.2022 v 11:30
Jedná se o přehledný graf
Pro rozkliknutí přenese na detail v úlohách
Datum automatického spuštění úlohy. Z důvodu předejití chybě není možné z této obrazovky provést ruční spuštění.


DMT GUI
Google
www.dmt-gui.cz/dictionaries
DMT Gui
Dashboard
Úlohy
Profily
Aplikace
v1.2.3
Slovníky
Zadej název...
Název
Datum vytvoření
Počet hodnot
Slovník 1
2022-03-03 10:30:00
53
Slovník 2
2022-03-03 10:30:00
16
Vyhledávání slovníku pomocí názvu. Vyhledání proběhne po změně hodnoty v textovém poli.
Dvojím klinutím na řádek se provede otevření slovníku pro editaci.
Umožňuje naimportovat slovník ze souboru tabulkového typu. Například csv.
Po nahrání souboru se otevře editor slovníku ve kterém bude možné dokončit import a provést uložení
Smazat
Nahrát ze souboru
Nový
Přejmenovat
Upravit


DMT GUI
Google
www.dmt-gui.cz/editor?id=123456
DMT Gui
Dashboard
Úlohy
Profily
Aplikace
v1.2.3
Editor profilu
Uložit
Uložit jako...
profil-migrace-1
<dmtMigrationProfile>
<MigrationConfiguration>
<Variables>
.
.
.
</dmtMigrationProfile>
1. <?xml version="1.0" encoding="utf-8"?>
2. <dmtMigrationProfile ...
3. .
4. .
5. .
6. .
7. .
8. .
9. .
10. .
11. .
12. .
13. .
14. </dmtMigrationProfile>
Formát: UTF-8, Formátování: Mezery
Problémy
Vstup
Výstup
2022-02-13 13:20:11: [Chyba] Chyba na řádku 3
2022-02-13 13:20:11: [Upozornění] Není deklarovaná proměnná @default_spisovy_znak
Vybrané elementy z profilu, které umožňují rychlý přesun na vybrané části profilu.
Veškeré informace potřebné k sestavení profilu a jeho ladění. Od informací po chyby.


DMT GUI
Google
www.dmt-gui.cz/logs
DMT Gui
Dashboard
Úlohy
Profily
Aplikace
v1.2.3
Výpis chyb
Export
Hledej
Date
Level
Message
Úloha
Profil
2022-03-03
INFO
Spuštění migrace dat
Migrační úloha 1
Profil 1
Provede vyhledávání všech chyb/informací, které budou podle zadaných parametrů k nalezení.
Detail zobrazí informace z vybraného řádku v přehledném samostaném okně. V případě více vybraných hodnot zobrazí první.
1
...
8
1000 z 7123 (vybráno 123)
Upravit
Detail

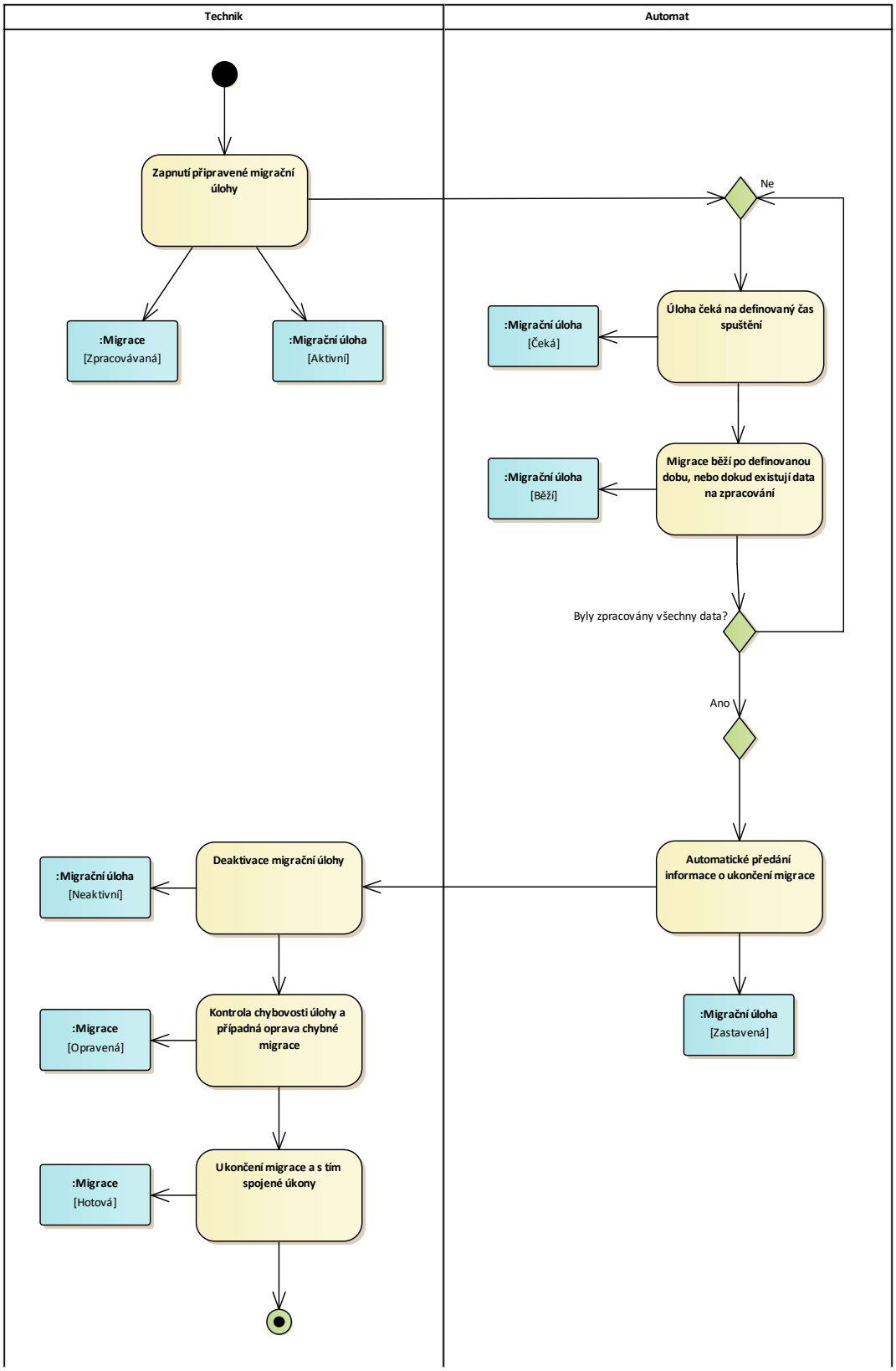

DMT GUI
Google
www.dmt-gui.cz/profiles
DMT Gui
Dashboard
Úlohy
Profily
Aplikace
v1.2.3
Seznam migračních profilů
Zadejte název…
Název
Úlohy
Profil 1
Migrační úloha 1 | Autoamtická migrační úloha 1
Profil 2
Migrační úloha 2
Profil 3
Migrační úloha 3
Vyhledávání profilu podle názvu
Úlohy: Seznam všech úloh, které používají daný profil. Kliknutím na daný název je možné otevřít profil napřímo.
Otevře vybraný profil v editoru profilů.
Uživateli se zobrazí dialog na nahrání profilů z disku a umožní import do aplikace.
Po nahrání do aplikace se uživateli zobrazí editor profilu, ve kterém bude možné dokončit import a profil uložit.
Smazat
Nahrát z disku
Nový
Upravit


DMT GUI
Google
www.dmt-gui.cz/jobs
DMT Gui
Dashboard
Úlohy
Profily
Aplikace
v1.2.3
Seznam úloh
Zadejte název...
Vyhledávání profilu podle názvu
Název
Stav
Plán
Ovládání
Migrační úloha 1
Běží
Není
Zastavit
Migrační úloha 2
Běží
Není
Zastavit
Automatická migrační úloha 1
Naplánovaná
21.3.2022 v 11:30
Spustit
Migrační úloha 3
Aktivní
Není
Spustit
Automatická migrační úloha 2
Neaktivní
Není
-
Je možné seřadit hodnoty podle vybraného sloupečku. Aktuálně seřazeno podle stavu.
Seznam všech vytvořených migračních úloh. Při rozkliknutí zobrazí bližší informací v "Detail vybrané úlohy".
Odstranit
Přidat

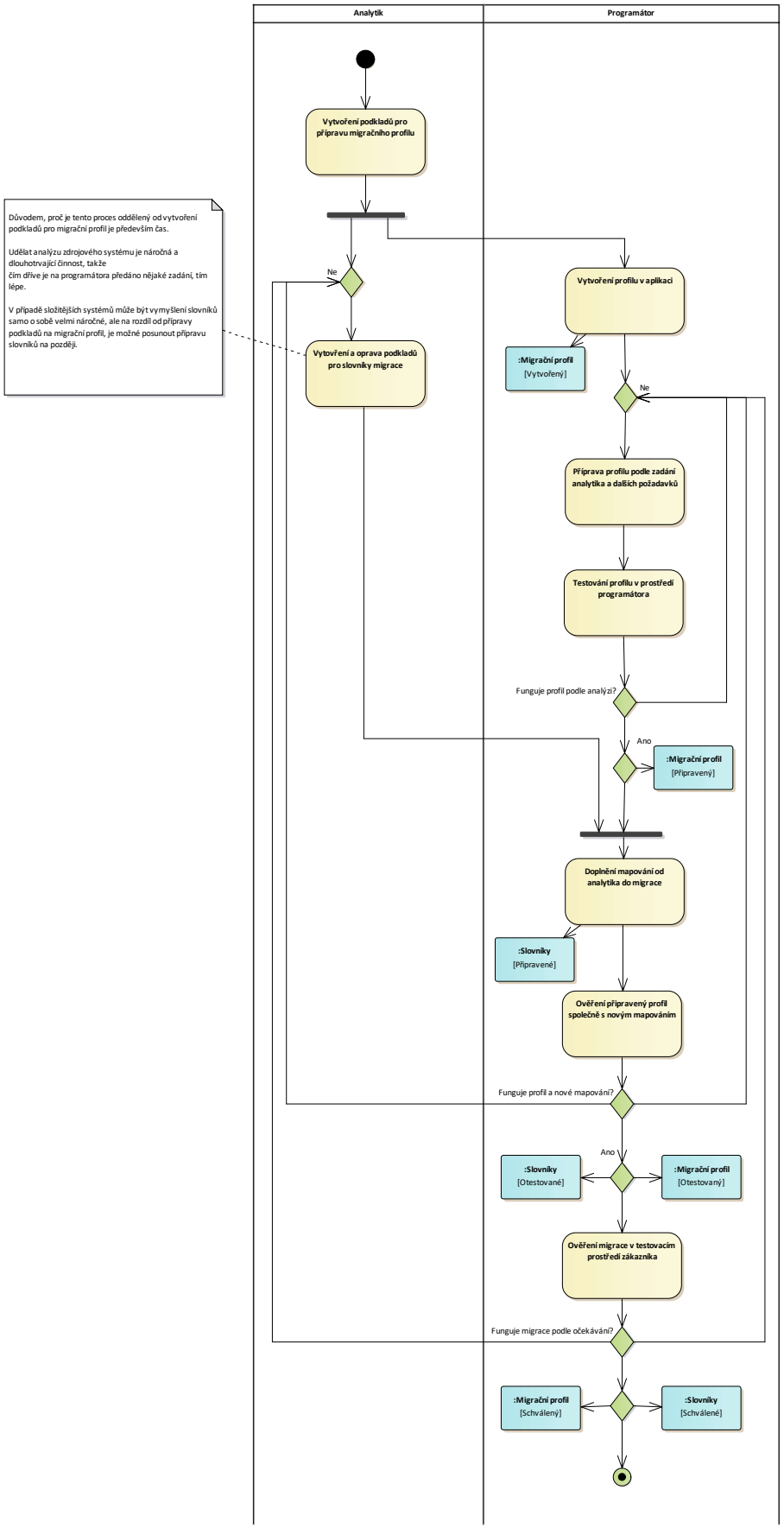
Příloha C

Diagramy procesů

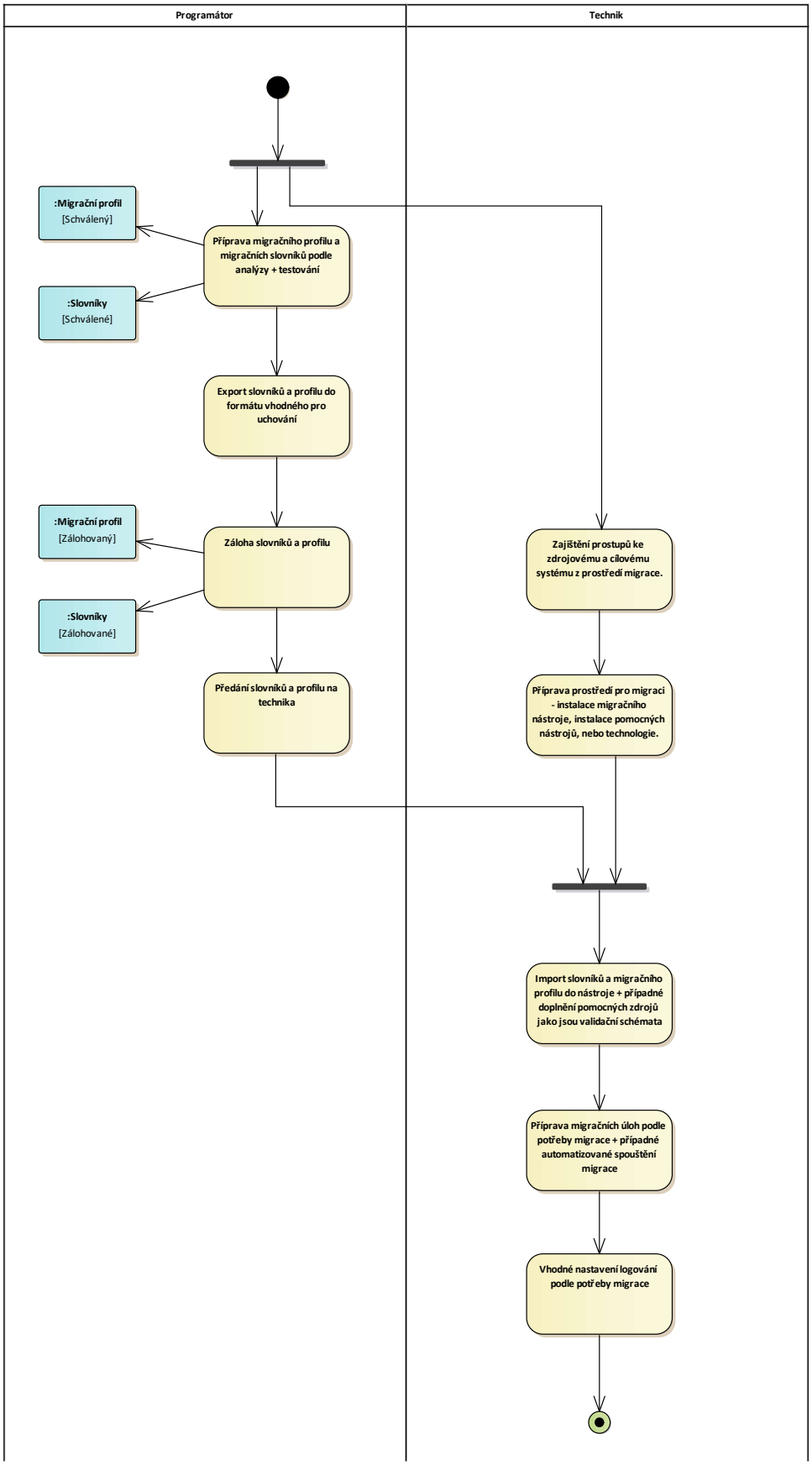
Obrázek C.1 Proces časované migrace



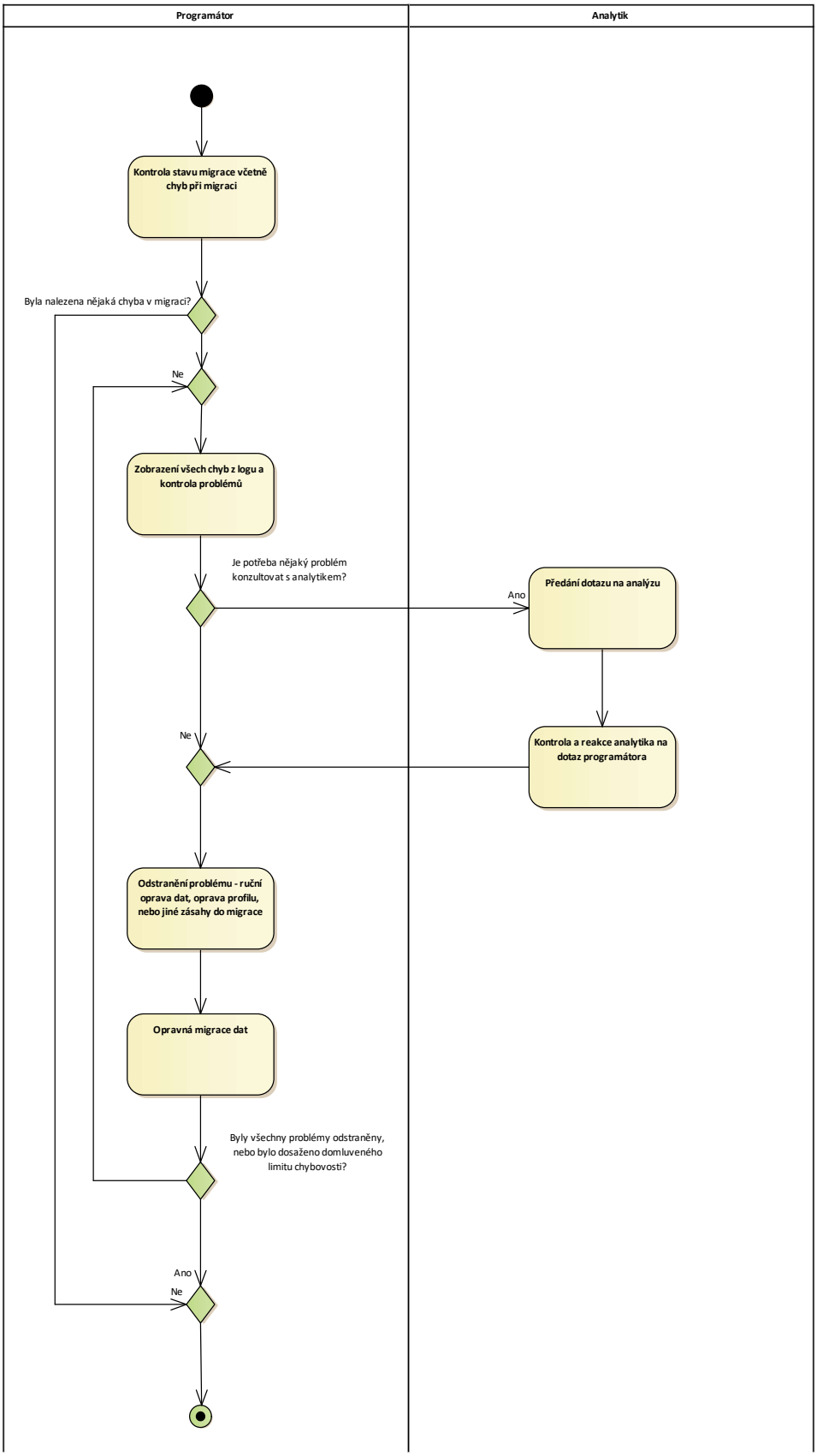
Obrázek C.2 Proces přípravy migračního profilu



Obrázek C.3 Proces přípravy nastavení



Obrázek C.4 Proces opravy dat



Příloha D

Instalační příručka

Nasazení aplikace se předpokládá dvěma způsoby: jako spustitelná konzolová aplikace, která poskytne grafické rozhraní přes prohlížeč, nebo jako instalovaná webová služba, která poskytuje stejné grafické rozhraní. Nasazení je popsané pro platformu Windows. Pro jiné platformy je možné použít jeden z návodů na adrese `https://docs.microsoft.com/cs-cz/aspnet/core/host-and-deploy/?view=aspnetcore-6.0`. V sekci D.2 je popsána konfigurace aplikace.

Instalace a konfigurace jsou popsány na souborech předaných na přiloženém médiu.

D.1 Instalace

Před spuštěním aplikace je potřeba nainstalovat běhové prostředí pro .NET 6 viz `https://dotnet.microsoft.com/en-us/download/dotnet/6.0`.

Instalace jako konzolová aplikace

Postup instalace a spuštění:

1. v systému vytvořit složku, ze které aplikace poběží
2. soubory ze složky `bin\release` včetně souboru `run.bat` zkopírovat do nové složky
3. podle potřeby nastavit aplikaci viz konfigurace v sekci D.2
4. pro zjednodušení je možné přidat odkaz na zkopírovaný spustitelný soubor `run.bat` na plochu
5. pomocí souboru `run.bat` je možné spustit aplikaci, provede se spuštění aplikace a otevře prohlížeč na adrese `http://localhost:5000`

Instalace jako Windows služba

Před instalací je potřeba nainstalovat aplikaci Powershell. Stačí ve verzi 5.1, protože ta přidává příkaz `New-Service`, který se používá k instalaci služeb. Optimálně je lepší použít verzi 6 či vyšší, protože ta navíc přidává funkci `Remove-Service` k odstranění nainstalovaných služeb.

Postup instalace je následující:

1. v systému vytvořit smysluplnou složku, ve které poběží webová aplikace
2. pokud bude webová služba spouštěná pod jiným uživatelem, než který ji vytvořil, tak je potřeba tomu uživateli přiřadit práva na zápis, čtení a spouštění složky

3. soubory ze složky `bin/release` zkopírovat do nové složky, soubor `run.bat` je možné vynechat
4. podle potřeby nastavit aplikaci viz konfigurace v sekci D.2
5. otevřít Powershell jako administrátor
6. pomocí příkazu D.1 vytvořit novou službu

Výpis kódu D.1 Vytvoření nové služby

```
New-Service `
        -Name "DMTGui" `
        -BinaryPathName "{Cesta do nove slozky}\DMTGui.Server.exe `
            --contentRoot {Cesta do nove slozky}" `
        -Credential "{domena\uzivatel pro spusteni aplikace}" `
        -Description "Migracni aplikace" `
        -StartupType Automatic
```

7. pomocí příkazu `Start-Service "DMTGui"` spustit webovou aplikaci
8. aplikaci je možné otevřít z prohlížeče na adrese `http://localhost:5000`
9. pokud je potřeba, je možné běh aplikace ovládat z manažeru pro služby Windows

Pokud dojde k chybě při startu nebo běhu webové aplikace, tak je možné najít logy v prohlížeči událostí pod názvem vytvořené služby. Další možnost je zapnout aplikaci pomocí spustitelného souboru `DMTGui.Server.exe` a podívat se na všechny chyby v konzoly aplikace.

D.2 Konfigurace

Konfigurace webového rozhraní

Konfigurace webové aplikace je umístěná v souboru `bin\release\appsettings.json`. V konfiguraci jsou vidět následující sekce:

Kestrel Definuje nastavení serveru hostující webovou aplikaci. Změnou `Endpoints.Http.Url` je možné definovat, na jakém portu bude server poslouchat a nabízet aplikaci. V případě změny je potřeba hodnotu opravit i v konfiguraci webového rozhraní a pokud se používá, tak i v souboru `run.bat`.

Logging Definuje nastavení logování ve webovém rozhraní.

App Definuje vlastní nastavení aplikace. Nastavení `Database` umožňuje definovat nastavení databáze, se kterou se pracuje. Pomocí `Database.ProviderType` je možné definovat konkrétní *provider*. Na výběr jsou možnosti „SQLite“, „SqlSever“ a „PostgreSQL“. Podle vybraného *provider* je potřeba nastavit validní *connection string* v `Database.ConnectionString`.

DMTLib Definuje nastavení pro migrační knihovnu.

Konfigurace webové aplikace

Konfigurace webové aplikace je umístěná v souboru `bin\release\wwwroot\appsettings.json`. V konfiguraci jsou vidět následující sekce:

Logging Definuje nastavení logování ve webovém rozhraní.

App Definuje vlastní nastavení aplikace. Nastavení `ServerUri` definuje adresu, na které komunikuje webová aplikace s webovým rozhraním. V případě této aplikace to znamená, že musí být nastavená stejná hodnota jako v nastavení webového rozhraní `Kestrel.Endpoints.Http.Url`.

Literatura

[1] Ministerstvo vnitra České Republiky, *Přichází DEPO* [Online], 1. 6. 2021 [cit. 17. 6. 2022]. Dostupné na: `https://www.mvcr.cz/clanek/prichazi-depo-snemovna-schvalila-balicek-podporujici-elektronizaci-verejne-spravy.aspx`

[2] ŠTOURAČOVÁ, Jiřina, *Archivnictví*. Brno: 2013. ISBN-13 978-80-210-6512-3

[3] Zákon č. 499/2004 Sb., o archivnictví a spisové službě a o změně některých zákonů. Platné od 23. 9. 2004

[4] Ministerstvo vnitra České republiky, *Národní standard pro elektronické systémy spisové služby* [Online], 3. 5. 2022 [cit. 17. 6. 2022]. Dostupné na: `https://www.mvcr.cz/clanek/narodni-standard-pro-elektronicke-systemy-spisove-sluzby.aspx`

[5] Ministerstvo vnitra České republiky, *Oznámení Ministerstva vnitra, kterým se zveřejňuje národní standard pro elektronické systémy spisové služby* [Online], 4. 6. 2017 [cit. 17. 6. 2022]. Dostupné na: `https://www.mvcr.cz/soubor/vestnik-mv-castka-c-57-2017.aspx`

[6] MORRIS, Johny, *Practical Data Migration, second edition*. England: 2012. ISBN 9781906124847

[7] ZHAO, Shirley, *What is ETL? (Extract, Transform, Load)* [Online], 20. 10. 2017 [cit. 17. 6. 2022]. Dostupné na: `https://www.edq.com/blog/what-is-etl-extract-transform-load/`

[8] BHANDARI, Rahul, LANDER, Rich, WENZEL, Maira, aj., *.NET 6 - Supported OS versions* [Online], 25. 5. 2022 [cit. 17. 6. 2022]. Dostupné na: `https://github.com/dotnet/core/blob/main/release-notes/6.0/supported-os.md`

[9] IBM Cloud Education, *Three–Tier Architecture* [Online], 28. 10. 2020 [cit. 17. 6. 2022]. Dostupné na: `https://www.ibm.com/cloud/learn/three-tier-architecture`

[10] DYKSTRA, Tom, KILLEEN, Sean, ABRAHAM, Isaac, aj., *What is .NET?* [Online], 10. 6. 2022 [cit. 17. 6. 2022]. Dostupné na: `https://docs.microsoft.com/en-us/dotnet/core/introduction`

[11] Stack Overflow, *Developer Survey: Web frameworks* [Online], 2021 [cit. 17. 6. 2022]. Dostupné na: `https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks`

[12] Tým WebAssembly, *Oficiální stránka pro WebAssembly* [Online], [cit. 17. 6. 2022]. Dostupné na: `https://webassembly.org/`

[13] GARDÓN, Diego Salinas, *Webassembly vs. JavaScript: How Do They Compare* [Online], 21. 3. 2022 [cit. 17. 6. 2022]. Dostupné na: https://snipcart.com/blog/webassembly-vs-javascript

[14] SMITH, Steve, JAIN, Tarun, PINE, David, aj., *Choose Between Traditional Web Apps and Single Page Apps (SPAs)* [Online], 14. 4. 2022 [cit. 17. 6. 2022]. Dostupné na: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps

[15] LATHAM, Luke, BERRY, Dina, SCOPEL, Fabio, aj., *ASP.NET Core Blazor* [Online], 9. 6. 2022 [cit. 17. 6. 2022]. Dostupné na: https://docs.microsoft.com/cs-cz/aspnet/core/blazor

[16] LAHMA, Marko, *QUARTZ: Open-source job scheduling system for .NET* [Online], [cit. 17. 6. 2022]. Dostupné na: https://www.quartz-scheduler.net/

[17] Codecademy Team, *What is REST?* [Online], [cit. 17. 6. 2022]. Dostupné na: https://www.codecademy.com/article/what-is-rest

[18] NOTLAR, Kendime, *ASP.NET MVC and Web API - Comparison of Async / Sync Actions* [Online], 30. 10. 2016 [cit. 17. 6. 2022]. Dostupné na: https://gokhansengun.com/asp-net-mvc-and-web-api-comparison-of-async-or-sync-actions

[19] AU-YEUNG, John a DONOVAN, Ryan, *Best practices for REST APIdesign* [Online], 2. 3. 2020 [cit. 17. 6. 2022]. Dostupné na: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/#h-name-collections-with-plural-nouns

[20] SHNEIDERMAN, Ben, PLAISANT, Catherine, COHEN, Maxine, JACOBS, Steven, ELMQVIST, Niklas a DIAKOPOULOS, Nicholas, *Designing the User Interface: Strategies for Effective Human-Computer Interaction, Sixth Edition*. 2016. ISBN-13 9780134380384

[21] Interaction Design Foundation, *Material Design* [Online], [cit. 17. 6. 2022]. Dostupné na: https://www.interaction-design.org/literature/topics/material-design

[22] SATPATHY, Lalatendu, *Designing better data tables for enterprise UX* [Online], 11. 7. 2020 [cit. 17. 6. 2022]. Dostupné na: https://uxdesign.cc/data-table-for-enterprise-ux-cb48fb9fdf1e

[23] ANDERSON, Rick, LATHAM, Luke, LARKIN, Kirk, aj., *ASP.NET Core Middleware* [Online], 3. 6. 2022 [cit. 17. 6. 2022]. Dostupné na: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware

[24] SPASOJEVIC, Marinko, *Global Error Handling in ASP.NET Core Web API* [Online], 23. 5. 2022 [cit. 17. 6. 2022]. Dostupné na: https://code-maze.com/global-error-handling-aspnetcore/

[25] SKEET, Jon, WAGNER, Bill, JAESCHKE, Rex, aj., *Annex D Documentation comments* [Online], 4. 5. 2022 [cit. 17. 6. 2022]. Dostupné na: https://github.com/dotnet/csharpstandard/blob/draft-v7/standard/documentation-comments.md

Obsah přiloženého média

- `readme.md` ... stručný popis obsahu média
- `bin` ... adresář se spustitelnou formou implementace
  - `release`
    - `DMTGui.Server.exe` ... spustitelná verze migrační aplikace
- `doc` ... adresář s dokumentací
  - `html` ... adresář s technickou dokumentací ve formátu html
    - `index.html` ... index stránka technické dokumentace
- `src`
  - `impl` ... zdrojové kódy implementace
  - `thesis` ... zdrojová forma práce ve formátu LaTeX
- `text` ... text práce
  - `thesis.pdf` ... text práce ve formátu PDF