FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Webová aplikace pro inteligentní mapování atributů v rámci datových integrací
Student:	Bc. Jan Toušek
Vedoucí:	Ing. Jan Hnízdil
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

V rámci integračních (ETL) procesů je jedním z nejnáročnějších úkolů vytvoření mapování mezi zdrojovými a cílovými entitami a atributy.

Cílem práce je vytvoření webové aplikace, která bude usnadňovat mapování atributů využitím podobnostních metrik na jména integrovaných entit a atributů.

Úkoly:

1. Provést rešerši již dostupných integračních (ETL) řešení
2. Provést rešerši metod vhodných pro porovnávání podobnosti textových řetězců
3. Navrhnout a implementovat webovou aplikaci s jednoduchým GUI s následujícími funkcionalitami
3a) Uživatel má možnost vytvořit připojení k datovým zdrojům
3b) Aplikace inteligentně navrhne mapování entit a jejich atributů mezi zdrojem a cílem
3c) Uživatel může návrh mapování potvrdit a uložit, nebo zamítnout a upravit
3d) Export vytvořeného mapování v navrženém formátu
3e) Webová aplikace bude umět mapování vykreslit pro potřeby dokumentace
4. Otestovat webovou aplikaci a její uživatelské rozhraní

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 2. prosince 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Webová aplikace pro inteligentní mapování atributů v rámci datových integrací

Bc. Jan Toušek

Katedra webového inženýrství
Vedoucí práce: Ing. Jan Hnízdil

5. května 2022

Poděkování

Děkuji přátelům a rodině za to, že mě podporovali po celou dobu studia a především při psaní této práce. Děkuji také vedoucímu práce Ing. Janu Hnízdilovi za vedení práce a jeho aktivní přístup.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 5. května 2022 . . . . . .

Odkaz na tuto práci

Toušek, Jan. *Webová aplikace pro inteligentní mapování atributů v rámci datových integrací*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022. Dostupný také z WWW: ⟨`http://site.example/thesis`⟩.

Abstrakt

Vývoj datových integrací dnes většinou vyžaduje od vývojáře manuální vytvoření mapování mezi zdrojovými a cílovými entitami a atributy. Rešeršní část této práce se zabývá existujícími nástroji používanými pro vývoj datových integrací. Důraz je kladen na jejich schopnost mapování navrhnout automaticky a také na grafické rozhraní, ve kterém je možné definovat mapování manuálně. Dále jsou popsány metody, které je možné využít pro nalezení podobnosti mezi databázovými schématy. Následující kapitoly popisují návrh a implementaci webové aplikace, která bude automaticky provádět mapování mezi vybranými schématy a navrhne jej vývojáři ke schválení, nebo úpravě.

Klíčová slova datová integrace, mapování entit, mapování atributů, podobnost databázových schémat, webová aplikace

Abstract

Developing data integrations today, in most cases, requires the developer to manually create mappings between source and target entities and attributes. The research part of this thesis discusses the existing tools used for developing data integrations. Is is focused on their ability to suggest mappings automatically and as well on the graphical interface in which mappings can be defined

manually. Furthemore, methods that can be used to find similarities between database schemas are described. The following chapters describe the design and implementation of a web application that will automatically perform the mapping between the selected schemas and propose it to the developer for approval or modification.

Keywords data integration, entity mapping, attribute maping, database schema similarity, web application

Obsah

Seznam obrázků

Seznam tabulek

Úvod

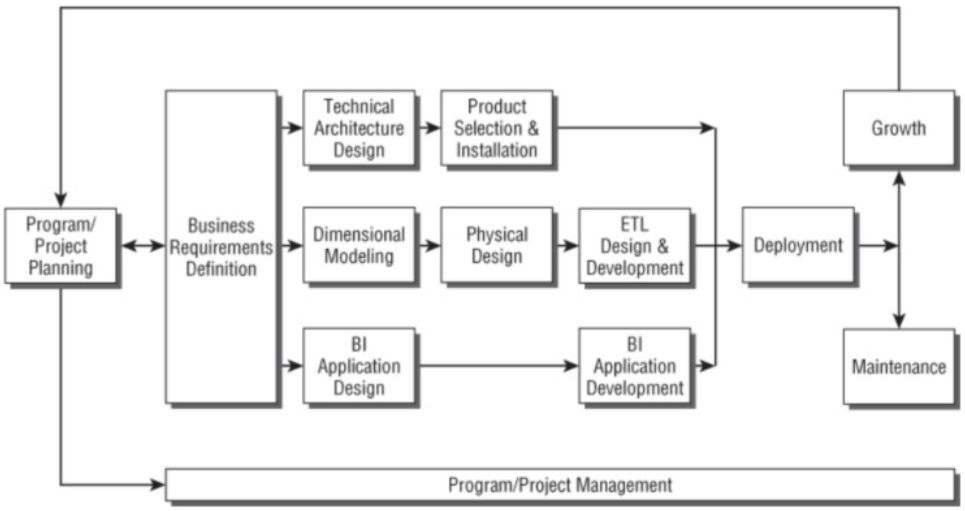
Dnešní společnosti pracují se stále se zvětšujícím objemem dat. Tato data jsou navíc často uložena v různých systémech. Mohou mezi ně patřit například systém pro správu zákazníků (CRM), systém pro plánování podnikových zdrojů (ERP), nebo systém pro řízení dodavatelského řetězce (SCM). Navíc bývá nad daty z těchto a dalších systémů vystaven datový sklad (DWH) pro potřeby business inteligence.

Kritériem při návrhu datového skladu je co nejvyšší přidaná hodnota pro danou organizaci. Z toho důvodu je potřebné, aby data v něm byla správně očištěna, propojena přes jednotlivé dimenze a aby aktuální data byla dostupná podle potřeby.

Možný postup při implementaci projektu datového skladu je znázorňuje Obrázek 0.1. Kimballův diagram životního cyklu business inteligence projektu znázorňuje úkoly potřebné k dokončení celého projektu. Nejvíce časově náročnou částí je návrh a implementace ETL systému, tedy procesu, který zajišťuje samotnou datovou integraci. Kimball [1] definuje 34 subsystémů, které jsou potřebné pro správné fungování celého řešení. Patří mezi ně mimo jiné systém pro extrakci dat, systém pro správu dimenzí, nebo systém pro plánování úloh. "Před zahájením návrhu ETL systému pro dimenzionální model by měl být dokončen logický návrh, vypracován vysokoúrovňový plán architektury a navrženo mapování mezi zdroji a cíli pro všechny datové prvky."[1]

Právě na zmíněné mapování mezi zdroji a cíli, je zaměřena tato diplomová práce. Jejím cílem je navrhnout a implementovat aplikaci, která urychlí proces vytvoření ETL procesu. Toho docílí tím, že automaticky navrhne mapování jednotlivých entit a atributů mezi zdrojovým a cílovým systémem na základě jejich podobnosti. Tím urychlí práci vývojáře a umožní mu zaměřit se na implementaci dalších systémů.

Tento systém poté bude možné použít nejen v oblasti datových skladů, ale v libovolném odvětví využívajícím datové integrace.



Obrázek 0.1: Kimballův diagram životního cyklu ETL projektu [1]

Kapitola 1

Mapování databázových schémat

Nedílnou součástí implementace integračního rešení je vytvoření mapování mezi zdroji a cíli. V současné době jej vývojáři musí vytvářet manuálně. Proces tvorby mapování od nich vyžaduje znalost sémantiky dat ve zdrojovém i cílovém systému a je časově náročný. Tato kapitola se zabývá rešerší postupů, které je možné uplatnit pro namapování databázových schémat bez nutnosti zásahu vývojáře.

1.1 Definice problému

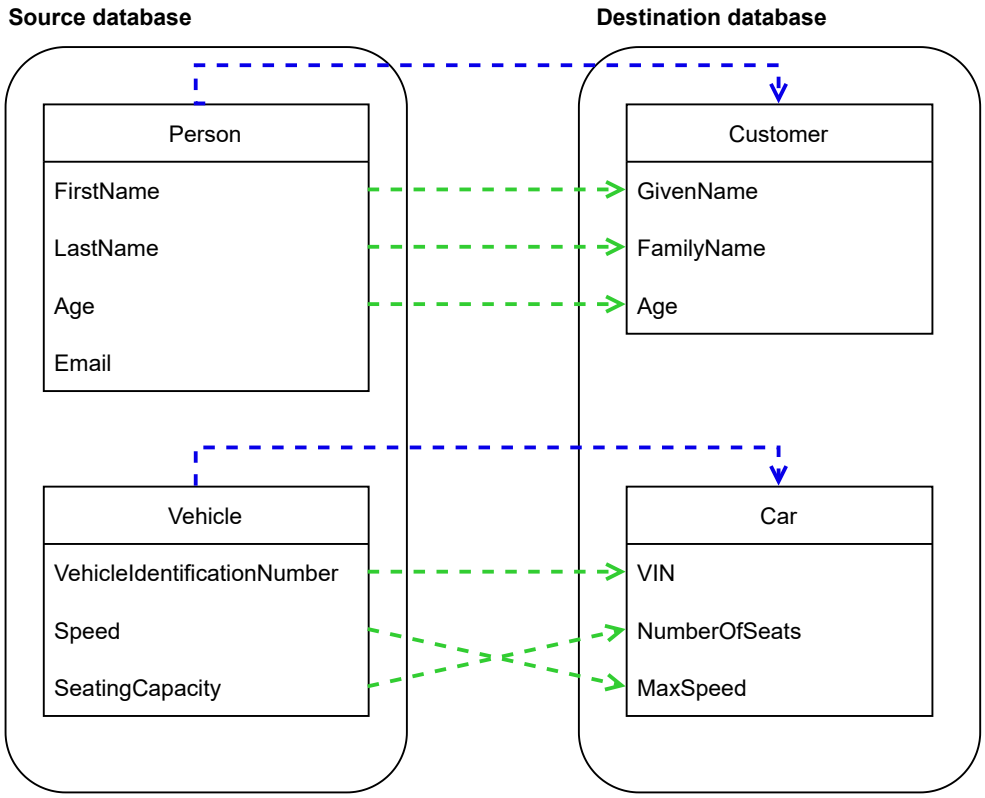
Mapování schémat je problém nalezení odpovídajících si prvků v různých schématech. Tento úkol je obtížné automatizovat, protože konkrétní sémantika dat je známa pouze návrhářům daného schématu a nemusí v něm být plně zachycena. Cílem je pro každý prvek ve schématechch S a T nalézt prvek ve druhém schématu, který mu odpovídá, pokud existuje. [2]

Příklad problému mapování dvou schémat znázorňuje Obrázek 1.1. Zdrojová databáze `Source` obsahuje tabulky `Person` a `Vehicle`. Ty je potřeba namapovat na Tabulky `Customer` a `Car` v cílové databázi `Destination`. Modrými šipkami je naznačeno mapování tabulek a zelenými šipkami mapování jejich atributů.

V příkladu můžeme vidět několik problémů, se kterými je možné se setkat při mapování. Názvy tabulek `Person` a `Customer` jsou odlišné, ale přesto mohou v nějakém kontextu reprezentovat stejné entity reálného světa, v tomto případě osoby.

Běžně používané atributy, jako je křestní jméno a příjmení mohou mít více pojmenování se stejným významem, v tomto případě `FirstName` a `GivenName`.

Některé atributy nemusejí mít ve zdroji, nebo v cíli odpovídající prvek, jako v tomto případě atribut `Age`. Další možné rozdíly budou popsány v sekci



Obrázek 1.1: Mapování databázových schémat

Heterogenita databázových schémat.

1.2 Mapování na základě podobnosti řetězců

Triviální způsob, jak vytvořit mapování mezi dvěma schématy je využití podobnosti textových řetězců. Tento způsob vychází z předpokladu, že napříč různými databázemi budou sémanticky totožné tabulky a atributy pojmenovány stejně, nebo alespoň podobně.

Zřejmou nevýhodou je, že entity a atributy nezvládne namapovat, pokud se jejich názvy budou významně lišit. To může být způsobeno například pojmenování synonymy, jiným jazykem, nebo využitím zkratek.

1.2.1 Podobnostní metriky

Níže budou popsány některé vybrané podobnostní metriky, které je možné využít pro aproximativní párování odpovídajících textových řetězců. Umožňují

nám určit, které textové řetězce jsou si podobné a které naopak zcela odlišné.

1.2.2 Editační vzdálenost

"Vzdálenost $d(x, y)$ mezi dvěma řetězci x a y je minimální cena posloupnosti operací, které transformují x na y (a ∞, pokud taková posloupnost neexistuje). Cena posloupnosti operací je suma cen jednotlivých operací."[3] Z toho vyplývá, že čím je vzdálenost menší, tím je podobnost větší. Pro stejné řetězce je vzdálenost nulová. Mezi zmíněné operace patří:

- vložení: vložení znaku a
- smazání: smazání znaku a
- substituce: nahrazení znaku a znakem b
- transpozice: výměna dvou sousedních znaků

Editační vzdálenost je často používaná metrika pro určení podobnosti textových řetězců. V závislosti na zvolených operacích jsou odvozeny různé varianty editační vzdálenosti.

Levenshteinova vzdálenost

Používá operace vložení, smazání a substituce. Všechny operace mají cenu 1.

Hammingova vzdálenost

Aplikuje operace substituce s cenou 1. Jinými slovy se jedná o počet znaků, ve kterých se řetězce liší.

Nejdelší společná podposloupost

Povoluje operace vkládání a mazání s cenou 1. Cílem je najít nejdelší podposloupnost, kterou obsahují oba porovnané řetězce. Cena je poté počet znaků, které nebyly napárovány.[1]

Jarova vzdálenost

Mějme dva řetězce s_1 a s_2, Jarova vzdálenost je vypočítána jako:

$$d_j = \begin{cases} 0 \\ \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_1|} + \frac{m-t}{|m|}) \end{cases}$$

Kde:

[1] Anglicky Longest Common Subsequence (LCS)

- m je počet stejných znaků na stejné pozici,
- $|s_1|$ je délka prvního slova,
- $|s_2|$ je délka druhého slova,
- t je 1/2 počtu zaměněných znaků (operace transpozice). [4]

1.2.3 Jaro-Winklerova vzdálenost

Jaro-Winklerova vzdálenost využívá Jarovu vzdálenost, ale dává větší váhu stejným znakům na začátku slov. Její hodnota je dána následující rovnicí:

$$d_w = d_j + (lp(1 - d_j)))$$

Kde p je prefixová škála s konstantní hodnotou (například 0.1) l je počet shodných znaků na začátku každého slova.

1.2.4 Jaccardův index

Také nazýván Jaccardův podobnostní koeficient je statistická metoda pro porovnání podobnosti množin. Používá podíl jejich průniku a sjednocení, neboli pro dvě konečné množiny A a B je hodnota Jaccardova indexu $J(A, B)$ definována:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

1.2.5 Kombinace metrik

V některých případech není možné určit nejlepší metriku, která bude nejlépe mít nejlepší výsledky při daném problému. Je tedy možné použít metrik více a výsledky agregovat (např. aritmetický průměr, minimum, maximum), případně využít nějakou jejich lineární kombinaci a umožnit tak určení váhy jednotlivých podobností.

1.3 Mapování s využitím metadat

Databázové tabulky a jejich příslušné atributy nejsou definovány pouze jmény. Můžeme k nim získat další užitečná metadata, díky kterým je možné částečně adresovat nevýhody předchozího způsobu založeného pouze na jménech.

Příklady metadat databázových sloupců:

- primární klíče
- cizí klíče

- datové typy
- velikost atributu
- unique omezení
- not null omezení
- textový popis

Tento způsob předpokládá, že zmíněná metadata si často budou odpovídat v různých schématech. Je pravděpodobné, že např. primární klíč tabulky ve zdrojové databázi bude odpovídat primárnímu klíči v cílové a podobně.

1.3.1 Podobnost metadat

Zmíněná metadata je možné reprezentovat jako vektor hodnot. Podobnost je poté vypočítána pomocí nekteré z metrik pro výpočet podobnosti vektorů, jako je například **Euklidova vzdálenost**, nebo **Cosinová podobnost**.

1.4 Mapování s využitím strojového učení

Některé existující systémy pro mapování schémat [5], [2] využívají strojové učení. Systém popsaný v [2] je založen na velkém korpusu schémat a jejich mapování v dané doméně. Tento korpus slouží k natrénování modelu, který vylepšuje mapování tím, že jsou porovnávaná schémata obohacena o znalost již existujících mapování. Navíc umožňuje využít statistiky jako jsou častá jména atributů, vztahy mezi tabulkami díky cizím klíčům a další.

Jiný přístup zvolili autoři [5]. Nevyužívají předem natrénovaný model, ale pro jeho vytvoření získávají statistická data z mapovaných schémat. Předpokládají dvě existující databáze, ve kterých se nacházejí data. Pro atributy v tabulkách obou databází provedou výpočet metrik, jako jsou průměr, průměrná délka, směrodatná odchylka, rozptyl a další. Na základě těchto statistik jsou atributy rozděleny do shluků (clustering) a nad těmi je provedeno hledání odpovídajících atributů.

1.5 Heterogenita databázových schémat

Tato sekce se zabývá různými faktory, které mohou hrát roli v odlišnosti různých databázových schémat. Těmito odlišnostmi se podrobně zabývá práce [6], ze které je v této sekci čerpáno. Autoři rozlišují konflikty na úrovni schématu a konflikty na úrovni dat.

1.5.1 Konflikty schématu

Konflikty tabulka – tabulka

Tyto konflikty jsou způsobeny různými reprezentacemi informací v tabulkách. Patří mezi ně:

- **Konflikty jmen tabulek** – např. stejné tabulky mají různá jména, nebo různé tabulky stejná jména.
- **Konflikty struktury** – např. pokud se liší počet atributů v jednotlivých tabulkách.
- **Konflikty omezení** – např. neodpovídající si primární klíče, cizí klíče a další omezení (constraints).
- **Konflikty M:N** - Tyto konflikty se mohou objevit, pokud jsou stejná data reprezentovány různým počtem tabulek například při využití dekompozice.

Konflikty atribut – atribut

Konflikty atribut – atribut jsou způsobeny různými definicemi sémanticky stej-ných atributů v různých databázích.

- **Konflikty jmen atributů** – podobně jako u tabulek jsou způsobeny různými jmény stejných atributů, nebo stejnými jmény stejných atributů.
- **Konflikty defaultních hodnot** – jsou způsobené různými definicemi defaultních hodnot daných atributů.
- **Konflikty omezení** – může jít o konflikty mezi datovými typy, nebo mezi integritními omezeními.
- **Konflikty M:N** – mohou nastat například při reprezentaci adresy, kde v jednom schématu je adresa dekomponována na `Číslo popisné`, `Ulice` a `Město`, zatímco ve druhém je spojena do jednoho atributu `Adresa`.

1.5.2 Konflikty dat

Konflikty dat autoři dělí na nevalidní data a různé reprezentace stejných dat.

Nevalidní data

Takto jsou nazývány hodnoty, které by měly být v obou databázích stejné, ale buď z důvodu špatného vložení, nebo z důvodu nesprávné synchronizace se data liší.

Různé reprezentace stejných dat

Příklady různých reprezentací:

- **Různé výrazy pro stejná data** – různá slova, řetězce, kódy.
- **Různé jednotky** – např. dny/týdny, metry/stopy
- **Různá přesnost** – různá velikost domény, např. zařazení do intervalu oproti exaktní hodnotě.

1.6 Konvence pojmenování databázových atributů

Přestože konvence jsou spíše doporučení pro vývojáře, jak mají databázové atributy a tabulky pojmenovávat, velká část se jimy řídí. Je tedy důležité tyto konvence zanalyzovat.

1.6.1 Obecné konvence

Doporučením pro pojmenování databázových objektů se věnuje například Joe Celko v knize SQL Programming Style. Vychází z normy ISO-11179-4. Datové objekty by měly mimo jiné:

1. být unikátně pojmenované
2. uvedeny v jednotném čísle
3. obsahovat pouze obecně známé zkratky
4. tabulky a další kolekce by měly být pojmenovány v množném čísle [7]

Prefixy

V oblasti databází je časté používání různých prefixů. Ikdyž je tato technika spíše na ústupu, je vhodné ji zmínit. Prefixy slouží například pro rozlišení různých databázových objektů.

- V_ - View
- SP_ - Stored Procedury
- T_ - Tabulky
- zkrácený název tabulky jako předpona jména sloupce, např. `CUS_NAME`

1.6.2 World Wide Importers (MSSQL)

World Wide Importers[2] je známá ukázková databáze pro Microsoft Sql Server. Používá následující konvence:

- jména tabulek jsou v množném čísle, např:
  - `Customers`
  - `Invoices`
- jména tabulek i atributů používají PascalCase, např:
  - `CustomerName`
  - `InvoiceName`
- nepoužívají se podtržítka, pomlčky, prefixy sloupců

1.6.3 Oracle

Jmenné konvence pro Oracle vycházejí z příručky pro vývojáře, která doporučuje:

- Názvy tabulek uvádět velkými písmeny.
- Slova oddělovat podtržítky.
- Používat množné číslo.
- Primární klíč pojmenovat – {`tabulka`}`_PK`. [8]

Příklady názvů tabulek: `PO_VENDORS`, `AS_LOOKUPS`

1.7 Různé jazyky databázových systémů

Tato sekce se věnuje otázce různých jazyků ve dvou porovnávaných databázích. Podobně jako v jiných odvětvích informačních technologií je velmi používaným jazykem angličtina, nicméně přesto mohou vývojáři pojmenovat databázové objekty v jiných jazycích.

Možným řešením tohoto problému je využití některého z existujících překladačů, jako jsou například Google Translator, nebo DeepL. Tyto překladače poskytují REST API, které umožňuje zadat cílový jazyk a překládaný text. Zdrojový jazyk nemusí být vyplněn a nástroj se jej pokusí rozpoznat automaticky. Použití takového API pro překlad bez znalosti zdrojového jazyka je naznačeno v ukázkách kódu 1 a 2.

[2] `https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/wide-world-importers`

```
POST /v2/translate?auth_key=[yourAuthKey]> HTTP/1.0
Host: api.deepl.com
User-Agent: YourApp
Accept: */*
Content-Length: [length]
Content-Type: application/x-www-form-urlencoded

auth_key=[yourAuthKey]&text=Hello, world&target_lang=DE
```

Ukázka kódu 1: Ukázkový dotaz na API DeepL [9]

```
{
  "translations": [{
    "detected_source_language":"EN",
        "text":"Hallo, Welt!"
  }]
}
```

Ukázka kódu 2: Ukázková odpověď API DeepL [9]

1.8 Existující systémy pro mapování

Přestože vzniklo mnoho článků v oblasti mapování databázových schémat, většina navrhovaných systémů zůstala na úrovni prototypu, nebo akademického projektu, který se nestal populární. Mezi výjimky patří systém COMA[3] vytvořený v roce 2002 na Lipské univerzitě.

COMA, neboli COmbination MAtch uživateli umožňuje vytvoření mapovací strategie v závislosti na problému. Uživatel může využívat implementované algoritmy a kombinovat je s pomocí poskytnutých kombinačních strategií. [10]

[3] https://sourceforge.net/projects/coma-ce/

Kapitola 2

Rešerše ETL nástrojů

Zkratka ETL z anglického Extract, Transform, Load je používána pro označení procesů, které zajišťují přenos dat mezi zdrojovým a cílovým systémem.

Tato kapitola se zabývá rešerší nástrojů, které jsou používány pro implementaci ETL procesů. Cílem je zanalyzovat jejich možnosti z hlediska automatického návrhu mapování a také jejich uživatelské rozhraní, které je pro vytvoření mapování využíváno.

2.1 ETL Nástroje

Na trhu existuje mnoho komerčních i Open source nástrojů zabývajících se ETL. Velmi používané jsou nástroje s grafickým rozhraním, v poslední době ale nabývají na popularitě i knihovny usnadňující vytváření ETL například programátorům v jazyce Python, nebo C#.

Autor knihy The data warehouse toolkit Ralph Kimball preferuje tradiční ETL nástroje. Tyto nástroje se podle Kimballa staly standardem v tomto odvětví díky jejich výhodám:

- Čtení dat z různých zdrojů – soubory, OLE DB, ODBS a další ovladače relačních databází.
- Obsahují funkcionality umožňující transformaci dat.
- Mají vlastnost sebe-dokumentace.
- Tvoří metadatový základ pro všechny kroky procesu.
- Umožňují správu verzí pro prostředí s více vývojáři a pro možnost obnovení předchozí verze.
- Obsahují pokročilou transformační logiku, například algoritmy fuzzy porovnávání, nebo deduplikace dat.

- Vyšší výkon systému při nižší úrovni odborných znalostí vývojářů – málo vývojářů SQL je skutečnými odborníky na to, jak relační databázi používat k manipulaci s extrémně velkými objemy dat.

- Pokročilé možnosti zpracování – např. paralelizace úloh a spouštění procesu na záložním stroji v případě výpadku. [1]

2.1.1 Nástroje s grafickým rozhraním

Jejich výhodou je jednoduchá konfigurace procesů, kterou zvládají i méně zkušení vývojáři. Ti mohou proces nakonfigurovat pomocí různých komponent, které jim umožňují definovat datové zdroje, datové cíle i jednotlivé transformace. K jejich použití často není nutná znalost programování.

Tyto nástroje mají ale i své nevýhody. Vlastnost sebe-dokumentace, kterou popisuje Kimball není ideální, protože pro její zobrazení bývá nutné otevřít daný projekt v příslušném nástroji. Jen toto otevření u větších projektů může trvat i desítky minut (například u SSIS).

Podobný problém souvisí s tvrzením, že ETL nástroj tvoří metadatový základ. Datové integrace, ve kterých je potřeba integrovat menší množství entit bývají jednoduché na konfiguraci a jejich zobrazení v grafickém nástroji je přehledné. Nicméně pokud je potřeba integrovat u větších projektů desítky, nebo stovky entit, začíná být orientace v projektu náročná. I při drobných úpravách, jako je přidání sloupce, nebo úprava datového typu poté vývojář musí procházet grafickými komponentami a hledat místo, kde má úpravu provést.

2.1.2 ETL knihovny

Některé společnosti a vývojáři namísto tradičních ETL nástrojů preferují využití knihoven, pomocí kterých datovou integraci naprogramují. Populární jsou například Python knihovny Apache Airflow[11], nebo Luigi[12]. Tento přístup vyžaduje od vývojářů znalost programování, nicméně při správném návrhu může poskytnout snažší orientaci ve velkých projektech.

2.1.3 Sledovaná kritéria

Cílem rešerše je zanalyzovat jednotlivé nástroje z pohledu mapování entit a atributů ze zdrojového systému na entity a atributy v cílovém systému. Vyhodnotit, jestli nástroj alespoň částečně umí tuto činnost udělat automaticky, nebo jestli vše musí udělat vývojář manuálně. Dalším cílem je ohodnotit uživatelské rozhraní pro mapování v jednotlivých nástrojích a inspirovat se z nich při návrhu vlastního grafického rozhraní pro mapování a zobrazení namapovaných atributů.

2.1.4 Kontext analýzy

Analýza nástrojů bude provedena s ohledem na nejpoužívanější případ užití datových integrací. Pro jednoduchost tedy bude pracováno s integracemi dvou databázových tabulek.

Pro příklady bude využita struktura databáze letišť OpenFlights.[13]

2.2 SQL Server Integration Services

Platforma SQL Server Integration Services umožňuje vytváření datových integrací a transformací. Poskytuje nástroje pro podniky, které řeší kopírování a stahování souborů, načítání datových skladů, čištění a vytěžování dat, ale také správu objektů SQL Serveru. [14]

2.2.1 Automatické mapování atributů

Pro vytvoření datové integrace uživatel nejprve vytvoří SSIS balíček. V rámci balíčku má poté možnost vytvořit tok dat. Tok dat vyžaduje datový zdroj a datový cíl. Zdroj a cíl jsou definovány pomocí správce připojení, kde uživatel typicky zadá údaje pro připojení k dané databázi. Poté již může vybrat zdrojovou a cílovou tabulku.

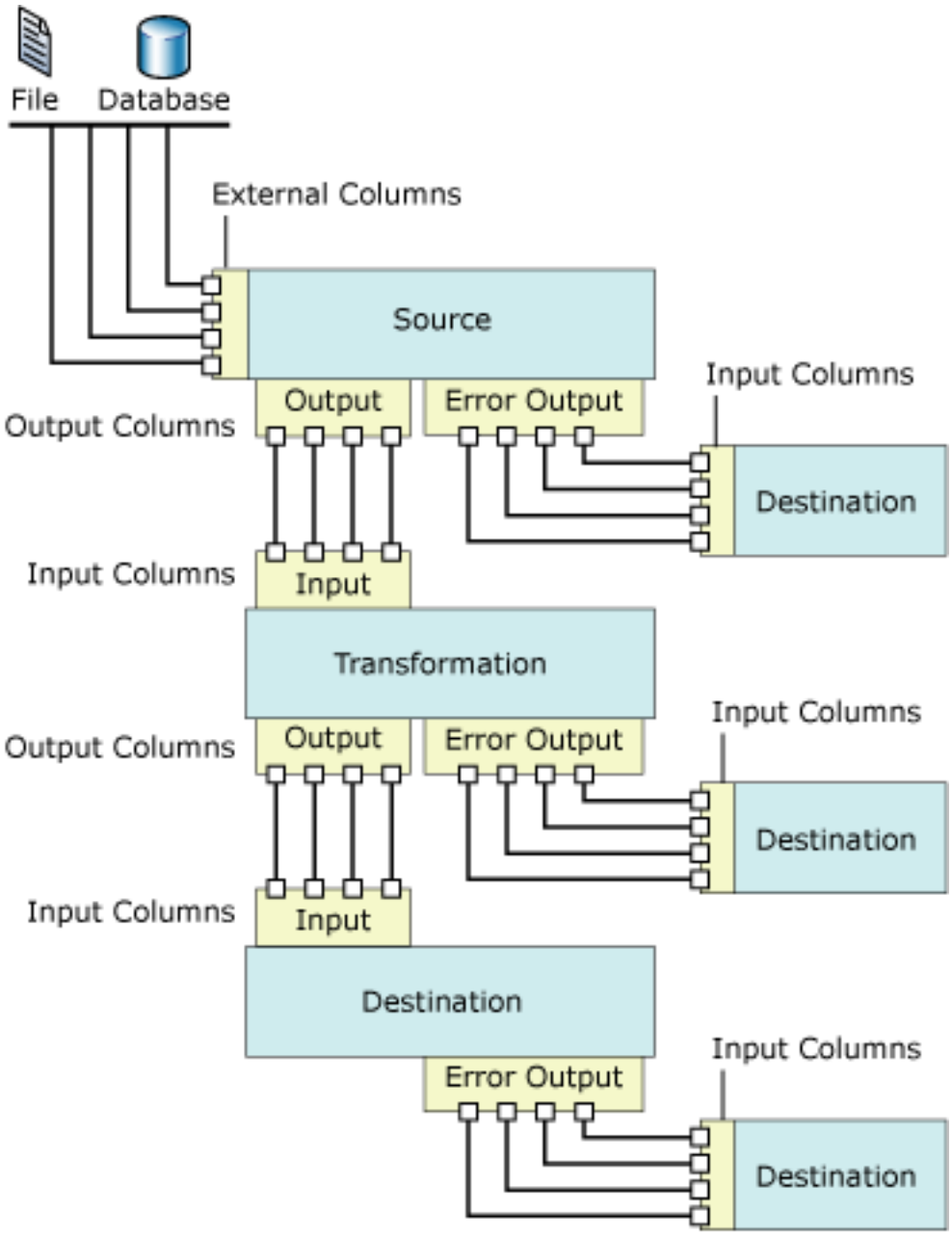
Schéma komponenty datového toku je zobrazeno na obrázků Obrázek 2.1. Struktura tabulky je načtena z datového zdroje v komponentě **Source**. Výstupní atributy komponenty pak uživatel mapuje na vstupní atributy komponenty **Transformation**, ve které má možnost definovat transformace. Nakonec pro-běhne mapování na vstup komponenty **Destination**, která zajistí uložení dat do cíle.

Nástroj dokáže automaticky propojit pouze atributy, které mají stejné jméno ve zdroji i v cíli, mohou se lišit pouze kapitalizací. Jakékoliv další odlišnosti musí vyřešit uživatel manuálně.

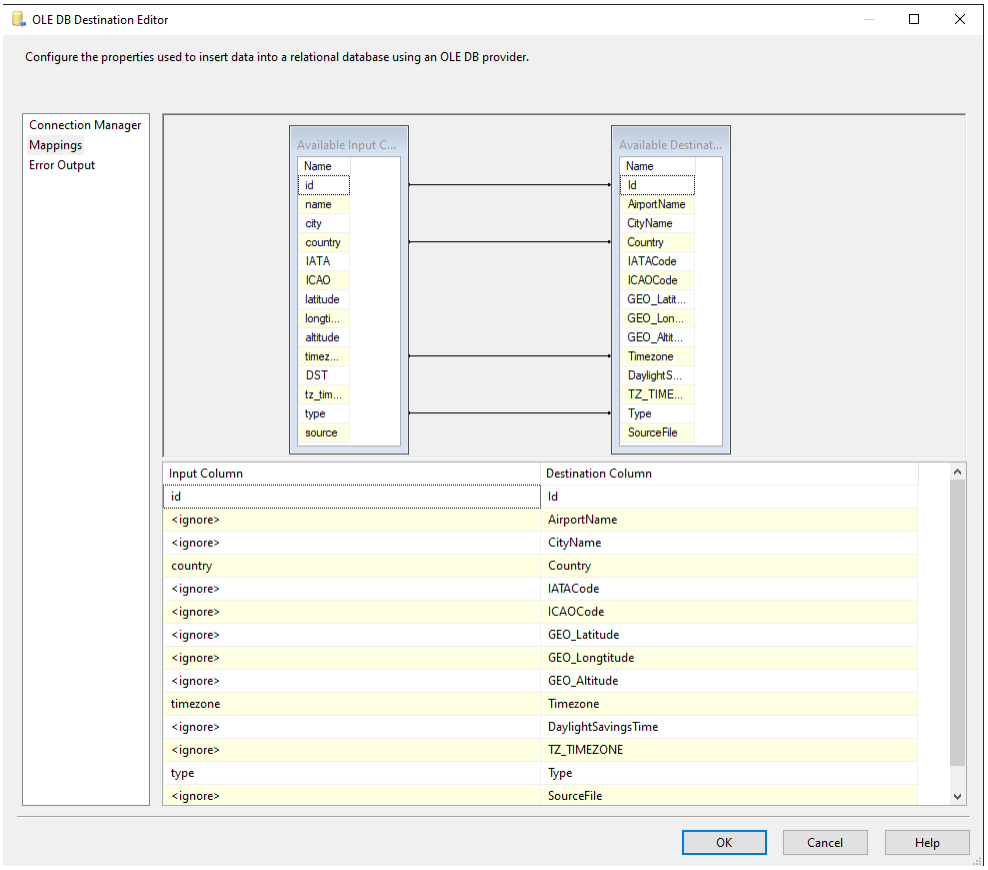
2.2.2 Grafické rozhraní pro mapování

Nástroj pro mapování, jak zobrazuje Obrázek 2.2 obsahuje dvě části. Horní polovina slouží k zobrazení provedeného namapování, kde jsou v jednom sloupci vstupní atributy a ve druhém výstupní. V dolní polovině obrazovky je seznam vstupních a výstupních sloupců. Vstupní sloupce nejsou specifikované a uživatel má možnost pomocí drop-down vybrat vstupní sloupec, který má být namapován na výstupní. Namapované sloupce jsou poté v horní části propojené šipkou.

Pokud tabulky obsahují větší množství atributů, než se vejde na obrazovku, je zobrazení mapování velmi nepřehledné, protože šipky se spojují do nejspodnějšího atributu. Grafické znázornění v horní části se pak stává nad-



Obrázek 2.1: Vstupy a výstupy komponent datového toku v SSIS [15]



Obrázek 2.2: Mapování atributů v SSIS

bytečným a pro uživatele je jednodušší porovnat seznam sloupců ve spodní části.

Pokud chce vývojář například integrovat entitu `Customers`, musí nejprve vybrat příslušnou zdrojovou a cílovou tabulku. Mapování entit je dáno vytvořením datových toků, jeho definice je tedy pouze na vývojáři. Toto mapování není vývojáři navrženo ani pro stejnojmenné tabulky.

2.3 Pentaho Data Integration (Kettle)

Po roce 2000 bylo na trhu málo kvalitních ETL nástrojů a téměř žádný nebyl Open source. Konzultant v oblasti business inteligence Matt Casters chtěl vytvořit nástroj, který bude mít:

- otevřený, čitelný metadatový formát (XML),
- otevřený, čitelný formát relačního úložiště,

- otevřené API,
- jednoduché nastavení,
- otevřený pro všechny druhy databází,
- grafické uživatelské rozhraní jednoduché na používání,
- možnost jednoduše přenášet data,
- možnost jednoduše konvertovat data z/do různých formátů. [16]

Vytvořil nástroj Kettle v jazyce Java a s využitím XML pro serializaci metadat. Poté co byl vydán jako open source, získal na popularitě a dodnes je používán například pro výukové účely. Nástroj byl později odkoupen a nyní je známý pod názvem Pentaho Data Integration.

2.3.1 Automatické mapování atributů

Mapování entit musí uživatel podobně jako v SSIS provést manuálně. Nástroj také automaticky propojí stejnojmenné atributy. Nicméně navíc obsahuje funkcionalitu **Guess**, která dokáže propojit i atributy, jejichž jména se liší, ale jsou si podobná. Tato funkcionalita je inspirací pro implementovaný nástroj automatického mapování.

2.3.2 Grafické rozhraní pro mapování

Grafické rozhraní pro mapování zobrazuje Obrázek 2.3. Rozhraní obsahuje tři sloupce. V prvním sloupci je seznam zdrojových sloupců, ve druhém seznam cílových sloupců a ve třetím je seznam vytvořených mapování. Pro vytvoření mapování musí uživatel označit jeden zdrojový a jeden cílový sloupec a přidat je pomocí tlačítka **Add**. Podobně lze mapování zrušit jeho výběrem a kliknutím na tlačítko **Delete**.

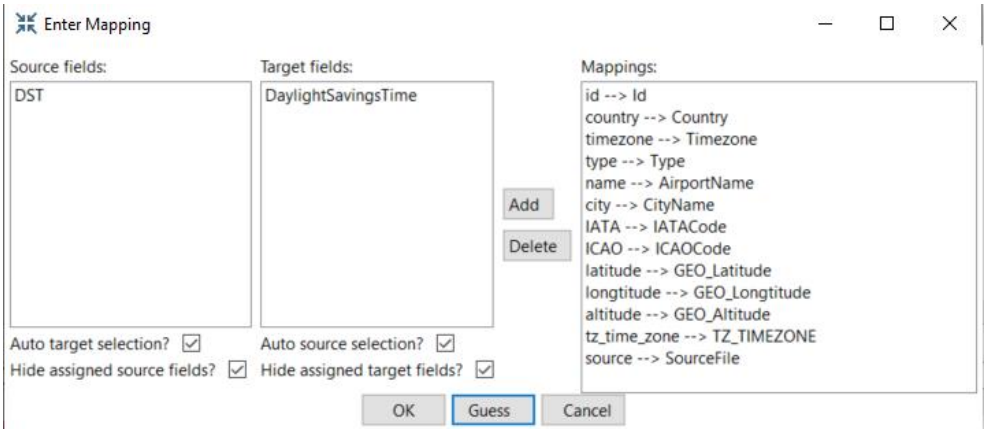
2.4 Azure Data Factory

Jedná se o cloudovou službu pro ETL a datové integrace, která umožňuje vytvářet procesy datových integrací a transformací ve velkém měřítku. [17]

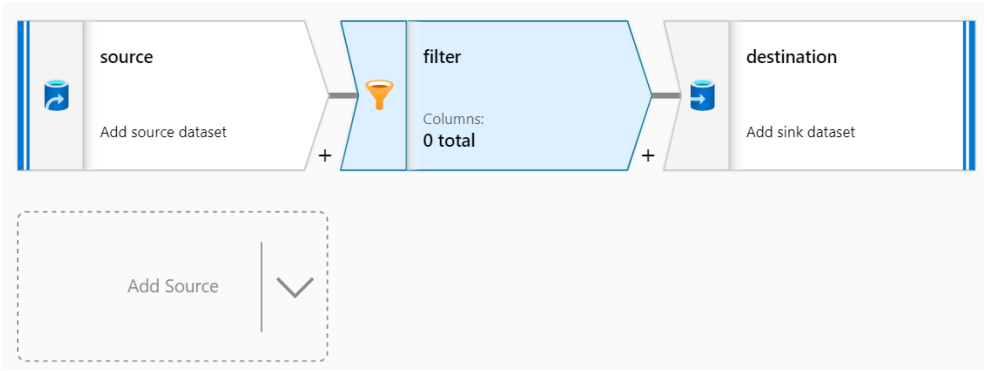
Nástroj umožňuje definovat integrační proces na základě metadat, nebo pomocí grafického rozhraní, ve kterém vývojář přidává potřebné komponenty a nastavuje jejich parametry.

2.4.1 Automatické mapování atributů

Podobně jako u předchozích nástrojů musí uživatel manuálně definovat zdroj a cíl. Nástroj automaticky namapuje pouze atributy, které se jmenují stejně a jsou case-sensitive (rozlišují velká a malá písmena).



Obrázek 2.3: Mapování atributů v Pentaho Data Integration



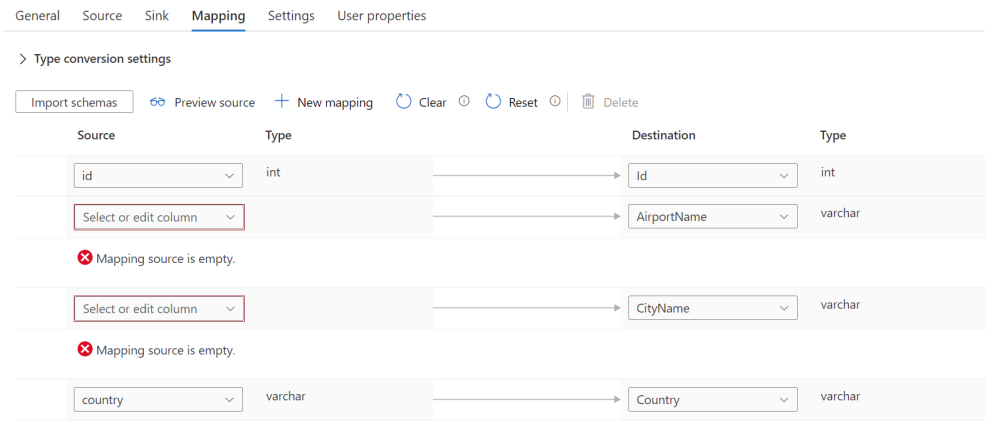
Obrázek 2.4: Rozhraní pro vytvoření datové integrace v Azure Data Factory

2.4.2 Grafické rozhraní pro mapování

Vytváření integračních procesů v Azure Data Factory pomocí grafického rozhraní umožňuje nástroj Azure Data Factory Studio. Rozhraní pro mapování v tomto nástroji zachycuje obrázek 2.5. Při výběru zdroje a cíle má uživatel k dispozici drop-down seznamy, ve kterých může vybrat správné atributy. Navíc rozhraní obsahuje informaci o datovém typu ve zdroji i cíli. Pokud chce vývojář namapovat novou dvojici, může ji přidat tlačítkem **New mapping**.

Nástroj Azure Data Factory navíc umožňuje definovat mapování pomocí JSON souboru. Příklad této konfigurace je v ukázce kódu 3. Konfigurace obsahuje název integrace, její typ, názvy připojení ke zdrojové a cílové databázi a poté seznam mapování zdrojového atributů na cílové.

Navíc kromě popsaného grafické rozhraní je možné integrační procesy



Obrázek 2.5: Mapování atributů v Azure Data Factory [18]

vytvářet programově. Pro tuto službu existují například knihovny v .NET[4] a Pythonu[5]. Další možností je používat přímo REST aplikační rozhraní.

2.5 ETLBox

ETLBox je .NET knihovna, která umožňuje implementaci ETL procesů například v jazyce C#. Příklad konfigurace integrace jedné entity je v ukázce kódu 4. Programátor vytvoří připojení ke zdrojové a cílové entitě a poté přidá mapování sloupců. Mapování je navrženo jako seznam dvojic jmen zdrojového a cílového sloupce.

2.5.1 Automatické mapování atributů

Tato ETL knihovna stejně jako dříve zmíněné Apache Airfloe a Luigi vyžadují, aby mapování nakonfiguroval programátor.

2.6 Shrnutí

V ideálním případě by měl být nástroj schopen ze schémat datového zdroje a cíle automaticky odvodit jak mapování mezi entitami, tak mezi jejich atributy. Potřebná metadata by mohl získat například pomocí připojovacího řetězce do databáze a následného načtení její struktury, nebo pomocí specifikace Open API, pokud by datovým zdrojem, nebo cílem bylo REST API.

Analyzované nástroje neposkytují plně automatizovanou funkcionalitu mapování. Nejvíce nabízí z tohoto pohledu nástroj Pentaho Data Integration

[4] `www.nuget.org/packages/Microsoft.Azure.Management.DataFactory`
[5] `pypi.org/project/azure-mgmt-datafactory/`

```
{
    "name": "CopyActivityTabularToTabular",
    "type": "Copy",
    "typeProperties": {
        "source": { "type": "SalesforceSource" },
        "sink": { "type": "SqlSink" },
        "translator": {
            "type": "TabularTranslator",
            "mappings": [
                {
                    "source": { "name": "Id" },
                    "sink": { "name": "CustomerID" }
                },
                {
                    "source": { "name": "Name" },
                    "sink": { "name": "LastName" }
                },
                {
                    "source": { "name": "LastModifiedDate" },
                    "sink": { "name": "ModifiedDate" }
                }
            ]
        }
    }
}
```

Ukázka kódu 3: Konfigurace mapování v Azure Data Factory pomocí JSON[18]

a jeho funkcionalita **Guess**, nicméně i tady je prostor pro vylepšení. Odhadnutí správného mapování by mohlo být provedeno automaticky. Navíc by nástroj mohl navrhnout mapování nejen atributů, ale i entit.

Méně uživatelsky přívětivý z analyzovaných produktů je v tomto ohledu nástroj SSIS. Uživatel musí mapování provádět ručně a při větším množství atributů je mapování nepřehledné.

Nástroj Azure Data Factory zobrazuje kromě jmen atributů i jejich datový typ ve zdroji a v cíli. Tato informace může být užitečná pro uživatele, který bude mapování konfigurovat.

ETL knihovny jsou z pohledu automatického mapování nejméně uživatelsky přívětivé, protože jej musí vytvořit programátor a nemá k dispozici grafické rozhraní. Nicméně případné propojení s navrhovaným nástrojem pro automatické mapování u nich bude relativně jednoduché.

```
var src = new DbSource(connectionManagerSrc, "TableSrc");
var dst = new DbDestination(connectionManagerDst, "TableDst");

source.ColumnMapping = new[]
{
    new ColumnMap {
        DbColumnName = "Id",
        PropertyName = "Number" },
    new ColumnMap {
        DbColumnName = "Col2",
        PropertyName = "Text" }
};

src.LinkTo(dst);
```

Ukázka kódu 4: Konfigurace mapování v knihovně ETLBox [19]

Kapitola 3

Analýza

Tato kapitola se zabývá analýzou, tedy prvním krokem při tvorbě softwarového řešení. Popisuje požadavky a funkcionality, které budou tvořit podklad pro následnou implementaci. Cílem je bližší specifikace zadání.

3.1 Funkční požadavky

Požadavky, které specifikují hlavní funkcionality navrhovaného systému.

3.1.1 F1 – Správa projektu

Aplikace bude umožňovat vytvořit a spravovat integrační projekty. Uživatel bude mít možnost projekt vytvořit, upravit a smazat.

3.1.2 F2 – Správa správců připojení

Aplikace bude umožňovat vytvořit a spravovat správce připojení. Uživatel bude mít možnost správce připojení vytvořit, upravit a smazat.

3.1.3 F3 – Vytvoření mapování

Při vytvoření, nebo úpravě projektu bude možné definovat mapování mezi dvěma databázovými systémy, k nimž bude přistupováno pomocí správců připojení. Mapování bude nejprve navrženo systémem automaticky a uživatel poté bude mít možnost navržené mapování potvrdit, upravit, nebo odstranit namapovanou dvojici z výstupu tak, aby nebyla součástí integrace.

3.1.4 F4 – Zobrazení mapování pro potřeby dokumentace

Webová aplikace bude umět mapování vykreslit pro potřeby dokumentace.

3.1.5 F5 – Export vytvořeného mapování

Mapování vytvořené v aplikaci bude možné exportovat v definovaném formátu.

3.2 Nefunkční požadavky

Požadavky, které definují omezení kladená na systém.

3.2.1 N1 – Webová aplikace

Systém bude vytvořen jako webová aplikace, dostupná přes webový prohlížeč. Aplikace bude zaměřena na přístup z počítačů.

3.2.2 N2 – Připojení k Microsoft SQL Server

Správce připojení bude umožňovat vytvoření připojení k databázím Microsoft SQL Server.

3.2.3 N3 – Formát exportovaného mapování

Export mapování bude možné provést ve formátu JSON.

3.2.4 N4 – Nasazení v dockeru

Webovou aplikaci bude možné nasadit jako docker kontejner.

3.2.5 N5 – Anglický jazyk

Aplikace bude lokalizována do angličtiny.

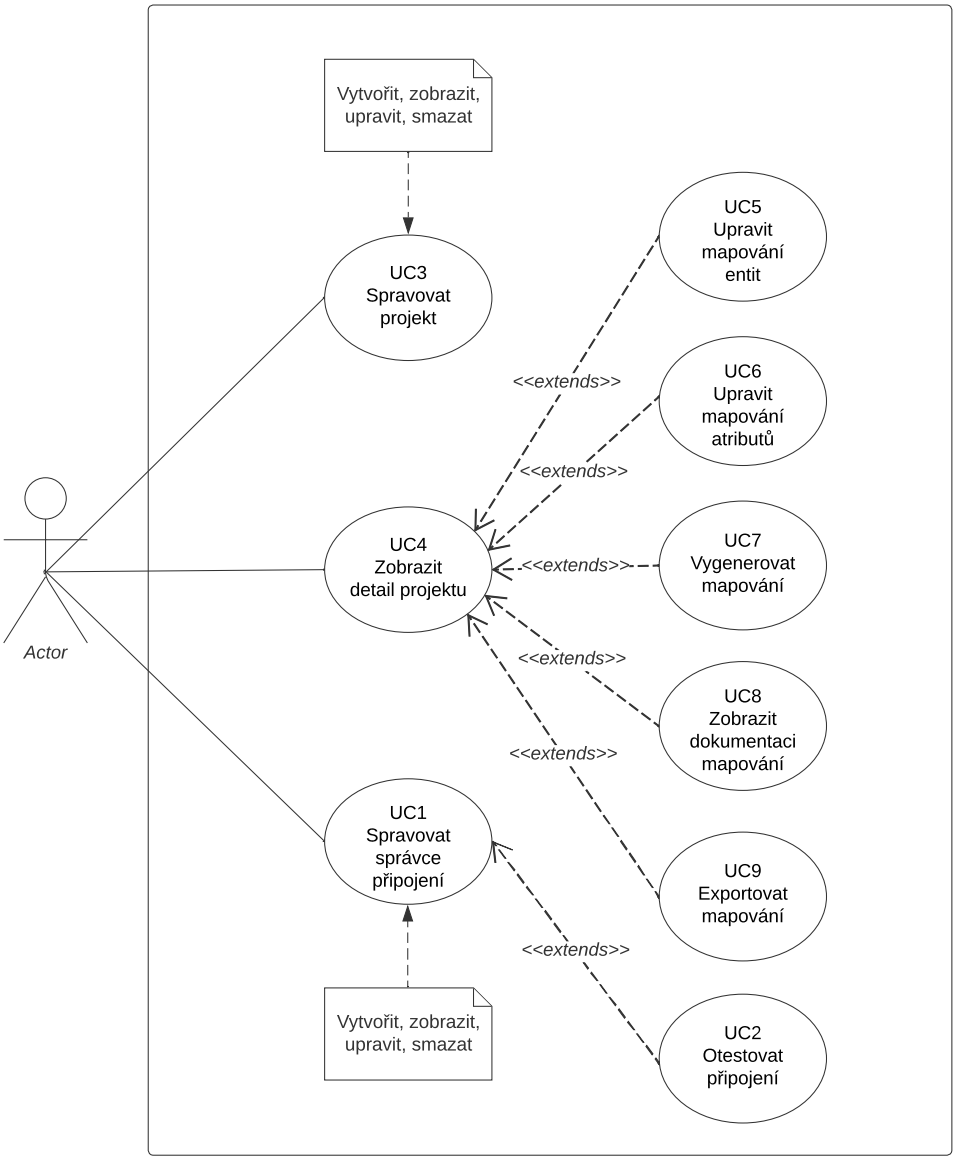
3.3 Případy užití (Use cases)

Na základě funkčních požadavků na aplikaci byly navrženy případy užití. Diagram případů užití je zobrazen na obrázku

3.3.1 Definice uživatele (Actor)

Navrhovaná aplikace bude sloužit především vývojářům v oblasti datových integrací. Jde o pokročilé uživatele, u kterých se předpokládá, že se v tomto odvětví pohybují a znají pojmy využívané v oboru.

Výše popsaný uživatel vystupuje v roli aktéra u všech následujících případů užití.



Obrázek 3.1: Diagram případů užití

3.3.2 UC1 – Spravovat správce připojení

Uživatel očekává možnost spravovat správce připojení. Správce připojení bude dále využíván aplikací pro načítání informací o schématech zdrojových a cílových databázových systémů. Aplikace umožňuje následující operace:

1. Vytvořit správce připojení zadáním připojovacího řetězce, nebo vyplněním přihlašovacích údajů.
2. Upravit správce připojení.
3. Zobrazit správce připojení.
4. Smazat správce připojení, po kliknutí na tlačítko smazání bude uživateli zobrazen potvrzující dialog.

3.3.3 UC2 – Otestovat připojení

Umožňuje uživateli při vytváření správce připojení otestovat zadané údaje.

1. Uživatel vyplní údaje pro připojení ke zvolené databázi.
2. Uživatel stiskne tlačítko Otestovat připojení.
3. Aplikace se pokusí připojení vytvořit a indikuje uživateli stav připojení.
4. V případě chyby připojení je uživateli zobrazena konkrétní chybová hláška.

3.3.4 UC3 – Spravovat projekt

Uživatel očekává možnost spravovat projekty. Aplikace umožňuje následující operace:

1. Vytvořit projekt.
2. Upravit projekt.
3. Zobrazit detail projektu.
4. Smazat projekt, po kliknutí na tlačítko smazání bude uživateli zobrazen potvrzující dialog.

Vytvoření projektu začíná kliknutím na tlačítko `Vytvořit projekt` na hlavní stránce aplikace.

1. Aplikace zobrazí formulář pro vyplnění parametrů potřebných pro vytvoření projektu.

2. Uživatel vyplní název projektu.
3. Pro datový zdroj i datový cíl vybere postupně z nabídnutého seznamu:
   - správce připojení,
   - databázová schémata,
   - typy databázových entit (tabulka, pohled),
   - entity, které mají být namapovány.
4. Uživatel uloží projekt stisknutím tlačítka `Uložit`.
5. Aplikace přesměruje uživatele na detail projektu.
6. Uživatel klikne na tlačítko `Spustit mapování`.
7. Aplikace spustí algoritmus mapování..
8. Aplikace do zobrazeného detailu projektu doplní navržené mapování včetně podobnosti jednotlivých entit a atributů.

3.3.5 UC4 – Zobrazit detail projektu

Případ užití začíná, když uživatel vytvoří nový projekt, nebo když vybere projekt ze seznamu již existujících.

1. Aplikace zobrazí detail projektu.
2. Aplikace zobrazí aktuální mapování mezi entitami.
3. Uživatel má možnost měnit stav návrhu mapování mezi stavy `V revizi` a `Potvrzeno`.
4. Uživatel má možnost vrátit se do kroku úpravy projektu a přidat, nebo odebrat entity.

3.3.6 UC5 – Úprava mapování entit

V zobrazení detailu projektu má uživatel možnost upravit mapování mezi entitami. Pro každou cílovou entitu může vybraz zdrojovou entitu z nabídnutého seznamu.

1. Uživatel vybere mapování, které chce upravit.
2. Uživatel klikne na rozbalovací seznam, který obsahuje zdrojové entity.
3. Uživatel vybere zdrojovou entitu, která má být namapována na cílovou.

3.3.7 UC6 – Úprava mapování atributů

Uživatel očekává možnost zobrazit a upravit mapování mezi atributy vybraných entit.

1. Uživatel klikne na detail vybrané entity.
2. Aplikace zobrazí seznam zdrojových a cílových atributů a jejich datových typů. Pokud již bylo provedeno mapování, zobrazí i jejich podobnost v procentech.
3. Uživatel má možnost změnit mapování mezi atributy.
4. Uživatel klikne na tlačítko Uložit, čímž uloží nastavení mapování atributů.
5. Aplikace uživatele přesměruje zpět na detail projektu.

3.3.8 UC7 – Vygenerovat mapování

Umožňuje uživateli automaticky vygenerovat navržené mapování mezi entitami a atributy na základě jejich podobnosti.

1. Na obrazovce detailu projektu uživatel stiskne tlačítko `Vygenerovat mapování`.
2. Aplikace spustí algoritmus mapování.
3. Aplikace do zobrazeného detailu projektu doplní navržené mapování, včetně zjištěné podobnosti jednotlivých entit a atributů.

3.3.9 UC8 – Zobrazit dokumentaci mapování

Pokud je již mapování vytvořeno, má uživatel možnost zobrazit v rámci detailu projektu namapované entity a atributy.

3.3.10 UC9 – Exportovat mapování

Uživatel očekává možnost exportu vytvořeného mapování.

1. Případ užití začíná, když uživatel dokončil mapování a provedl potřevné úpravy.
2. Uživatel Uloží vytvoření projekt stisknutím tlačítka `Uložit`
3. Uživatel stiskne tlačítko `Exportovat`.
4. Aplikace zahájí stahování souboru, ve kterém je serializováno vytvořené mapování ve formátu JSON.

	Případy užití								
Požadavky	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
F1			+	+					
F2	+	+							
F3					+	+	+		
F4								+	
F5									+

Tabulka 3.1: Tabulka pokrytí požadavků případy užití

3.4 Pokrytí požadavků

Tabulka 3.1 slouží ke kontrole splnění všech funkčních požadavků. V tabulce můžeme vidět, že každý řádek tabulky, který odpovídá příslušnému funkčnímu požadavku je pokryt nejméně jedním případem užití.

3.5 Algoritmus mapování

Tato sekce se zabývá algoritmem, který bude aplikace využívat pro nalezení mapování mezi entitami a atributy. Vychází z metod popsaných v kapitole 1.

3.5.1 Definice problému

Hlavním problémem, kterým se tato práce zabývá je nalezení co nejlepšího mapování mezi zdrojovými a cílovými entitami. Podproblémem pak je nalezení nejlepšího mapování mezi jednotlivými atributy daných entit.

Obecně tedy máme množinu $S = \{s_1, ...s_n\}$ zdrojových entit/atributů a množinu $D = \{d_1, ..., d_m\}$ cílových entit/atributů. Dále předpokládáme existenci matice podobnosti $M \in \mathbb{R}^{n,m}$, která obsahuje ke každému přiřazení atributu z S k atributu z D jeho podobnost. Pro prvky matice platí $M^{i,j} \in < 0, 1 >$, kde 0 reprezentuje žádnou podobnost a 1 je maximální podobnost.

Nyní uvažujme stejně velké množiny S a D a matici $M \in \mathbb{R}^{n,m}$. Algoritmus řešící tento problém hrubou silou by potřeboval vygenerovat všechny možná přiřazení z S do D. Prvnímu prvku z S lze přiřadit prvek z D n způsoby, druhý $(n-1)$ způsoby atd. Celkový počet možností je:

$$n(n-1)(n-2)...1 = n!$$

Pokud jsou množiny různě velké, např $|S| = n$, $|D| = k$ a $k \leq n$, odpovídá počet možností počtu k-členných variací z n prvků, tedy:

$$V(k,n) = \frac{n!}{(n-k)!}$$

Časová složitost algoritmu je tedy $O(n!)$.

3.5.2 Problém lineárního přiřazení

Problém popsaný v sekci 3.5.1 je variantou problému lineárního přiřazení, anglicky Linear Assignment Problem (LAP). Ten může být popsán následovně.

Mějme n jedinců ($i = 1, ..., n$) a n činností ($j = 1, ...n$). Dále mějme cenovou matici $R = (r_{ij})$, kde r_{ij} jsou kladná celá čísla pro všechna i a j. Přiřazení se skládá z výběru činnosti j_i pro každého jedince i tak, že žádná činnost není přiřazena dvou jedincům a zároveň každá činnost je někomu přiřazena. Přiřazení tedy tvoří permutaci

$$\begin{pmatrix} 1 & 2 & \dots & n \\ j_1 & j_2 & \dots & j_n \end{pmatrix}$$

čísel $1, 2, ..., n$.

Hledáme takovou maximální (nebo minimální) sumu $r_1 j_1 + r_2 j_2 + r_n j_n$. [20]

3.5.3 Kuhn-Munkresův algoritmus

Pro problém lineárního přiřazení existuje polynomiální Kuhn-Munkresův algoritmus, také nazývaný Maďarský algoritmus. Algoritmus byl původně navržen s časovou složitostí $O(n^4)$, ale později vylepšen pro dosažení složitosti $O(n^3)$.

Hlavní myšlenkou algoritmu je, že pro podobnostní matici platí následující tvrzení. Odečtení nejmenšího prvku sloupce od každého prvku v daném sloupci a odečtení nejmenšího prvku v řádku od každého prvku v řádku nezmění výsledné optimální přiřazení. Touto operací vznikne v každém řádku a každém sloupci alespoň jeden nulový prvek. V dalším kroku algoritmus hledá co nejmenší počet čar, kterými je možné pokrýt nulové prvky. Pokud je k pokrytí potřeba n čar, algoritmus nalezl optimální přiřazení, jinak pokračuje ve vytváření nulových prvků v matici.

3.6 Vytvoření podobnostní matice

Algoritmus nalezení mapování, popsaný v sekci 3.5 předpokládá existenci podobnostní matice, která definuje podobnost mezi jednotlivými prvky dvou množin. Tato sekce se zabývá vytvořením zmíněné matice.

3.6.1 Podobnostní matice atributů

Mějmě dvě tabulky, mezi kterými chceme vytvořit mapování jejich atributů. Výpočtem podobnosti každé dvojice atributů vznikne matice podobností. K výpočtu podobnosti využijeme data o jednotlivých atributech, která jsou v rámci Sql serveru k dispozici. Patří mezi ně:

1. názvy sloupců,

2. kompatibilita datových typů,

3. další metadata (PK, FK, Unique, ...).

Pro porovnání názvů sloupců bude využita Levehnsteinova vzdálenost. Datové typy budou porovnány na základě matice definující jejich kompatibilitu, tedy jestli je možné data ve zdrojovém atributu konvertovat na cílová. Další dostupná metadata budou uložena do vektoru pro oba porovnávané sloupce a jejich podobnost bude určena Cosinovou podobností.

Každá ze tří zmíněných metod vytvoří jednu podobnostní matici. Tyto matice agregujeme do jedné a tu následně využijeme jako výslednou podobnostní matici sloupců.

3.6.2 Podobnostní matice tabulek

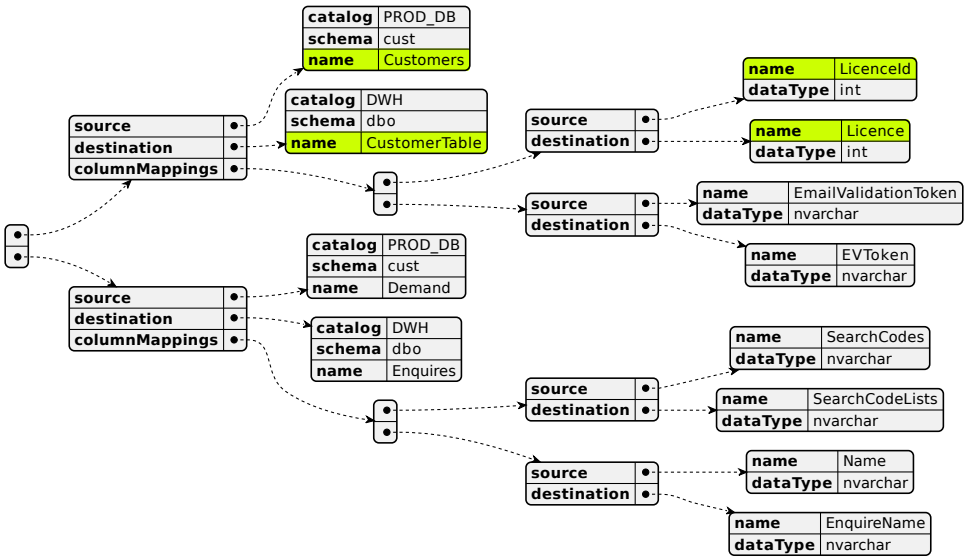
Podobně jako v případě sloupců potřebujeme i pro tabulky vypočítat jejich podobnostní matici. Ta je v případě tabulek určena podobností jejich názvů, ale také podobností jejich sloupců. Podobnost sloupců získáme na základě mapování sloupců, které vrátí Munkresův algoritmus pro jejich podobnostní matici.

Nad podobnostní matici tabulek je opět spuštěn Munkresův algoritmus a na základě jeho výstupu je navrženo mapování mezi tabulkami.

3.7 Návrh serializace metadat

Funkční požadavek F5 – Export vytvořeného mapování požaduje možnost exportovat vytvořené mapování ve formátu JSON. Schéma struktury souboru je vykresleno na obrázku 3.2. Soubor obsahuje JSON seznam mapování entit, kde každé mapování má zdrojovo entitu, cílovou entitu a dále seznam mapování sloupců. Mapování sloupců opět obsahuje objekt reprezentující zdrojový a cílový atribut.

Na obrázku je zvýrazněný příklad, kde entita `Customers` ve zdrojovém systému má být integrována na entitu `CustomerTable` v cílovém systému. Zdrojový sloupec `LicenceId` je namapován na cílový sloupec `Licence`.



Obrázek 3.2: Struktura serializovaného mapování v JSON

Kapitola 4

Návrh

4.1 Návrh uživatelského rozhraní

Kvalitní návrh uživatelského rozhraní hraje důležitou roli z pohledu user experience. Tato sekce je věnována jeho návrhu tak, aby aplikace byla co nejvíce uživatelsky přívětivá.

4.1.1 Wireframe prototyp

Wireframe prototyp je používán pro návrh rozložení prvků na obrazovce. Zobrazuje rozložení stránky a obsahu. V této práci je wireframe prototyp použit pro získání první zpětné vazby od potenciálních uživatelů. Díky tomu je možné případné nedostatky odstranit již v této fázi návrhu. Případné úpravy jsou tak časově podstatně méně náročné, než úpravy na základě testování funkčního prototypu, nebo hotové aplikace. Umožňuje také lépe si představit jednotlivé případy užití.

4.1.2 Skupina testerů

Jako testeři v této fázi návrhu byli zvoleni vývojáři v oblasti datových integrací. Náplní jejich práce je správa a implementace integračních řešení. V současné době využívají mimo jiné nástroje představené v sekci 2.1 a cílem navrhovaného software je urychlení jejich práce.

4.1.3 Scénář testování

Pro testery byl vytvořen následující testovací scénář. Testeři měli k dispozici wireframe prototyp a formou diskuze mohli k návrhu dávat zpětnou vazbu.

1. Vytvořit správce připojení.
2. Vyplnit parametry správce připojení.

3. Otestovat, jestli je připojení funkční.
4. Vytvořit nový mapovací projekt.
5. Vyplnit parametry projektu.
6. Uložit vybrané parametry.
7. Spustit mapování.
8. Potvrdit vybrané mapování a zamítnout vybrané mapování.
9. Potvrdit mapování všech entit, která mají konfidenční skóre větší, než 90 %.
10. Otevřít detail entity.
11. Upravit mapování atributů.
12. Doplnit manuálně mapování atributů, které nebylo navrženo automaticky.
13. Uložit změny v mapování.
14. Uložit projekt.
15. Exportovat mapování.
16. Upravit vytvořený projekt.
17. Smazat projekt.
18. Smazat správce připojení.

4.1.4 Výsledný prototyp

Testeři měli k dispozici wireframe prototyp a plnili úkoly podle vytvořeného scénáře. Počáteční verze prototypu byla iterativně upravována na základě jejich zpětné vazby. Výsledný prototyp je přiložen jako Příloha B.

4.1.5 User experience a heuristická evaluace

User experience se věnuje interakcím uživatele se systémem. Heuristická evaluace se provádí s cílem zvýšení použitelnosti systému. Nielsen identifikuje 7 nejdůležitějších faktorů, které zlepšují použitelnost systému. [21] V této sekci budou popsány kroky k vyhovění těmto požadavkům.

Viditelnost stavu systému

Uživatel by měl být informován o aktuálním dění.

Tento faktor se týká zejména funkcionality F3 – Vytvoření mapování, u které se předpokládá, že bude trvat delší dobu. Systém by měl uživatele informovat o aktuálním stavua průběhu operace.

Shoda mezi systémem a reálným světem

Systém by měl používat jazyk, fráze a koncepty kterým uživatel rozumí.

Uživateli aplikace budou vývojáři datových integrací. Jde o zkušené uživatele s dobrou znalostí prostředí informačních technologií, zejména v oboru datových integrací. Těmto uživatelům by aplikace měla být přizpůsobena.

Kontrola a svoboda uživatele

Uživatelé mohou nějakou akci provést omylem. Systém by jim měl umožnit zrušit danou akci a vrátit se do předchozího bodu bez nutnosti celý proces opakovat od začátku.

Aplikace bude umožňovat vrátit se do předchozího kroku pomocí tlačítka `Zpět`. Například při vytváření projektu.

Konzistence a standardy

Systém by měl dodržovat konvence a standardy běžné v daném odvětví.

Aplikace bude používat názvosloví shodné s dalšími produkty v odvětví datových integrací. Díky tomu bude pro nové uživatele snažší se naučit aplikaci používat.

Prevence chyb

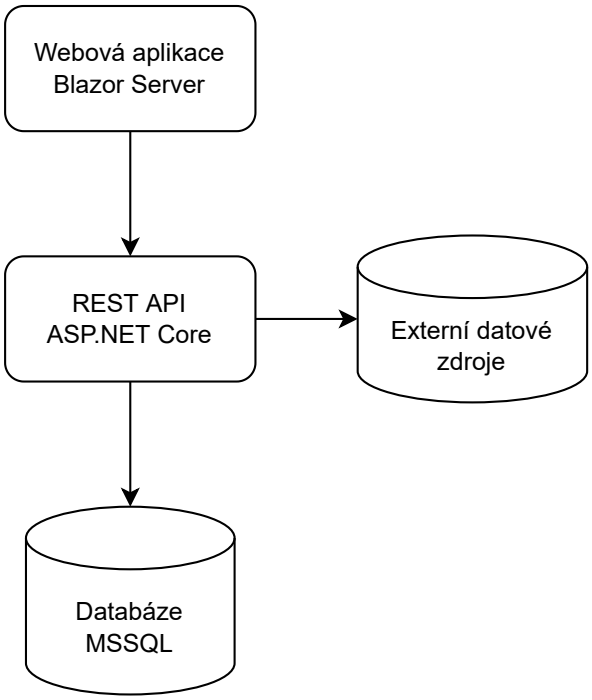
Systém by měl být navržen tak, aby minimalizoval počet chyb, které může uživatel udělat. Může jít o překliknutí, nebo chybu.

Funkcionality F1 – Správa projektu a F2 – Správa správců připojení budou při mazání obsahovat potvrzující dialog, aby nedošlo ke smazání projektu, nebo správce připojení omylem.

Spíše rozpoznávání než vzpomínání

Snaha minimalizovat zátěž uživatelovi paměti tím, že prvky a akce budou pro uživatele viditelné.

Rozhraní pro mapování by mělo zobrazovat všechny potřebné informace na jednom místě, tak aby je uživatel nemusel hledat.



Obrázek 4.1: Návrh architektury systému

Flexibilita a efektivita použití

Usnadnění používání pro zkušené uživatele, možnost přizpůsobení systému.

Pro zkušené uživatele je možné implementovat klávesové zkratky pro rychlejší pohyb v rámci aplikace.

4.2 Architektura a technologie

Výběr technologií vychází z nefunkčních požadavků, popsaných v sekci 3.2. Zejména tedy požadavků N1 – Webová aplikace a N4 – Nasazení v dockeru.

Pro implementaci aplikace byla zvolena vícevrstvá architektura. Obrázek 4.1 znázorňuje schéma navržené architektury. Jednotlivé části aplikace budou implementovány v ekosystému **.NET Core**.

4.2.1 Prezentační vrstva

Pro implementaci prezentační vrstvy byla zvolena technologie **ASP.NET Core Blazor**[6], nasazena jako **Blazor Server**. Aplikace je spuštěna na serveru v rámci ASP.NET Core aplikace. Aktualizace uživatelského rozhraní na

[6] `https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0`

klientovi, zpracování událostí a volání JavaScriptových funkcí je zajištěno pomocí SignalR spojení mezi klientem a serverem. [22] Samotné stránky aplikace budou implementovány jako Razor komponenty.

4.2.2 Logická vrstva

Při využití technologie Blazor Server je možné přistupovat k datové vrstvě napřímo, protože její serverová a klientská část jsou oddělené. Nicméně z důvodu oddělení zodpovědnosti je pro logickou vrstvu aplikace zvoleno samostatné REST API. API bude vytvořeno v technologii **ASP.NET Core** a bude zajišťovat business logiku aplikace.

Díky tomuto oddělení bude možné v budoucnu případně změnit model klientské aplikace na Blazor WebAssembly. Výhodou poté může být snížení záteže serveru tím, že se výpočet přenese na klientské aplikace. [22]

Využití těchto .NET technologií umožňuje implementaci prezentační i logické vrstvy ve stejném programovacím jazyce, kterým je C#. To umožňuje například sdílet společné knihovny mezi backendem a frontendem.

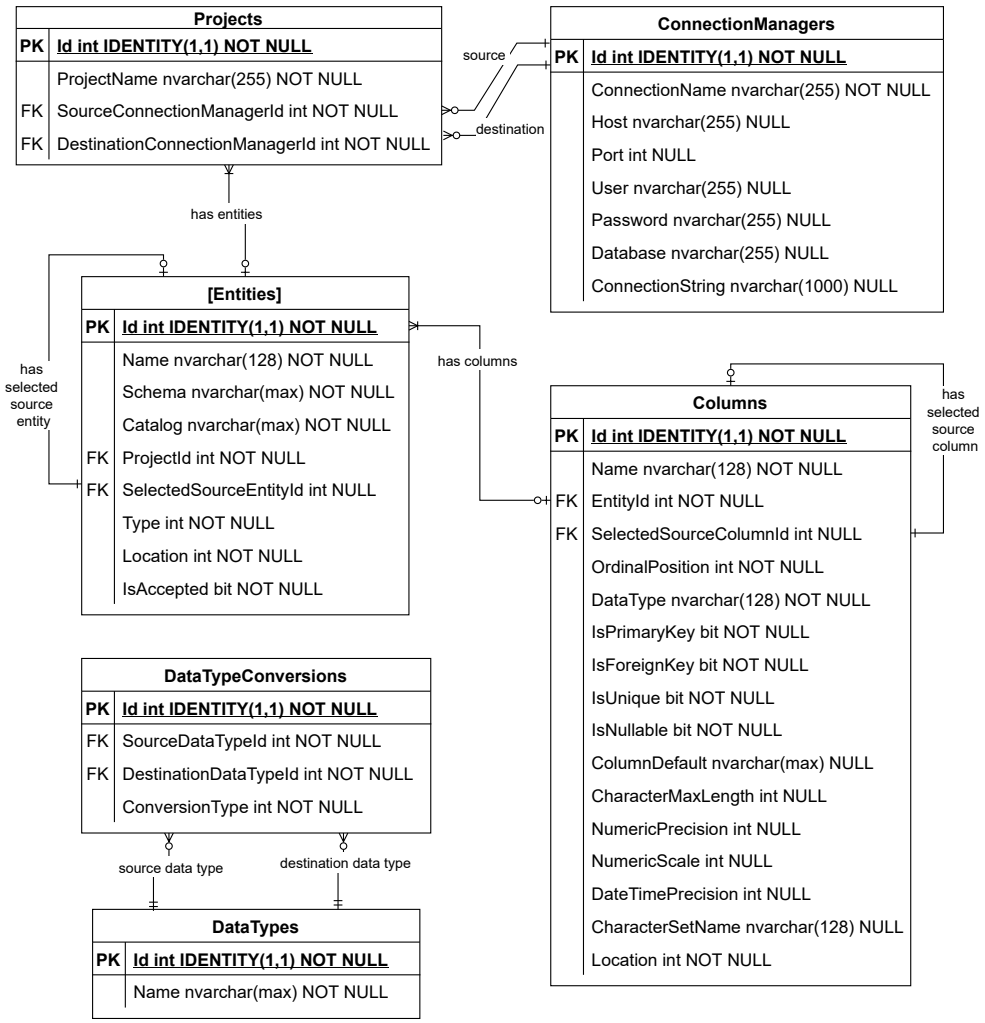
4.2.3 Datová vrstva

Jako perzistentní datové uložiště pro ukládání vytvořených projektů a jejich konfigurací byl zvolen databázový systém **Microsoft SQL Server**.

Schéma navržené databáze je zobrazeno na obrázku 4.2.

4.2.4 Externí datové zdroje

Po vytvoření správců připojení bude aplikace přistupovat k dalším externím datovým zdrojům definovaným uživatelem.



Obrázek 4.2: Schéma databáze

Kapitola 5

Implementace

5.1 Struktura vývojového řešení

Při vývoji v platformě .NET jsou Projekty organizovány v rámci řešení (Solution). Struktura projektu, včetně závislostí mezi projekty je vykreslena na obrázku 5.1. Řešení obsahuje následující projekty.

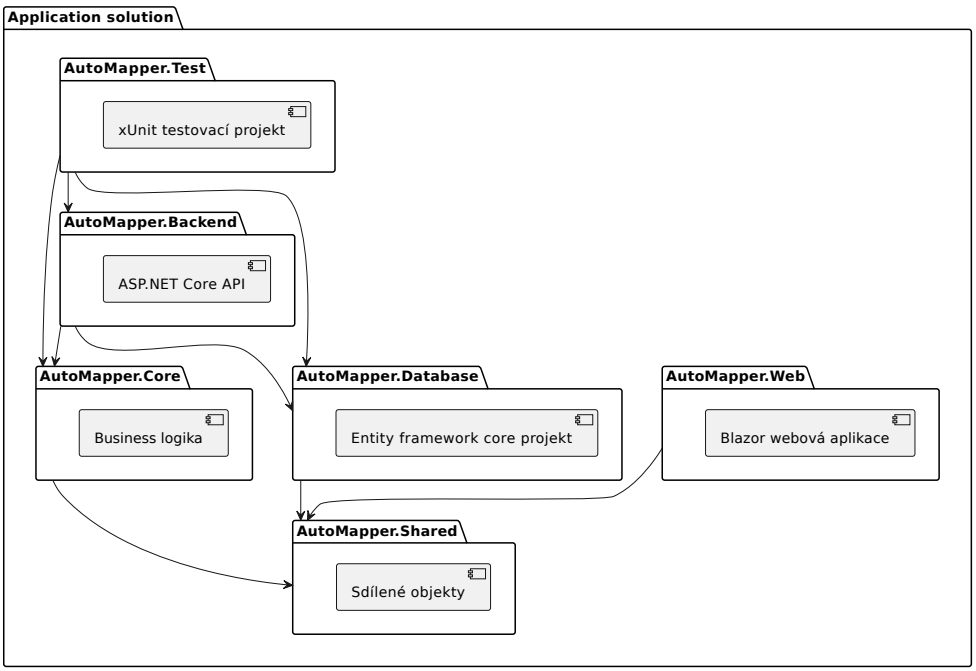
- **AutoMapper.Backend** – Backendové REST API v ASP.NET Core.
- **AutoMapper.Database** – Obsahuje funkcionalitu zajišťující přístup do databáze.
- **AutoMapper.Core** – Business logika aplikace, algoritmus mapování a načítání databázových schémat.
- **AutoMapper.Test** – Jednotkové a integrační testy.
- **AutoMapper.Web** – Blazor webová aplikace.
- **AutoMapper.Shared** – Knihovna obsahující Sdílené doménové třídy.

5.1.1 Přístup k aplikační databázi

Projekt **Automapper.Database** obsahuje databázový kontext v **Entity framework Core**[7], který je využíván pro přístup do databáze, ale také pro její vytváření, aktualizaci a inicializaci.

Pro vytvoření struktury databáze je použit **Code First** přístup. Veškeré databázové entity jsou vytvořeny pomocí doménových tříd, z nich jsou poté odvozeny tabulky a další objekty v databázi. Ke změnám struktury databáze jsou využívány Entity framework migrace. S těmi je možné pracovat například přes dotnet CLI.

[7] `https://docs.microsoft.com/en-us/ef/core/`



Obrázek 5.1: Návrh architektury systému

Konfigurace databázových entit

Entity Framework Core využívá konvence k sestavení modelu na základě entitních tříd. Konfiguraci je možné přidat, nebo upravit a tím přepsat použité konvence. [23]

V případě složitějších vztahů, jako je tomu u například u entity `ConnectionManager`, která obsahuje dvě vazby typu 1:N na entitu `Project`, je ale potřeba nastavení provést explicitně.

Explicitní konfigurace je definována pomocí tříd, které implementují interface `IEntityTypeConfiguration<T>` a metodu `Configure`.

V metodě `OnModelCreating` databázového kontextu je poté možné aplikovat všechny konfigurace implementující zmíněný interface voláním:

```
builder.ApplyConfigurationsFromAssembly(...);
```

5.2 Čtení metadat databáze

Schéma databáze, ze které chce uživatel načíst objekty pro mapování je získáno za pomoci třídy `SqlConnection`. [8]. Její metoda `GetSchema` umožňuje

[8] `https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/retrieving-database-schema-information`

přistupovat ke kolekcím, které umožňují jednoduché čtení metadat databáze.

Parametr `collectionName` specifikuje název dané kolekce.Kolekce **Tables** obsahuje tabulek a pohledů. Metadata týkající se sloupců včetně jejich vazby na příslušnou tabulku jsou dostupná v kolekci **Columns**.

Přestože existují kolekce **Foreign Keys**, **Indexes** a **IndexColumns**, data v nich dostupná nejsou dostatečná pro získání informací o primárních a cizích klíčích. Kolekce **Foreign Keys** obsahuje pouze informaci o tabulce ve které se klíč nachází, ale žádnou informaci o referencované tabulce.

5.2.1 INFORMATION_SCHEMA

Kolekce dostupné v rámci třídy `SqlConnection` zpřístupňují pouze několik vybraných kolekcí z `INFORMATION_SCHEMA`.

Toto schéma je jedním ze zdrojů metadat o objektech v SQL Serveru. Je navrženo podle ISO standardu, který definuje `INFORMATION_SCHEMA`. [24]

Zmíněný standard implementují i další databázové systémy, jako například Oracle a MySql. Z důvodu možné integrace dalších databázových systémů v budoucnu bylo zvoleno `INFORMATION_SCHEMA` jako zdroj metadat.

5.2.2 Načtení omezení

Schéma využitých pohledů z `INFORMATION_SCHEMA` je zobrazeno na obrázku Obrázek 5.2. Jednotlivé pohledy neobsahují přímo primární klíče, ale vazby mezi nimy lze odvodit na základě atributů, které obsahují.

TABLES

Řádek je jednoznačně identifikován trojicí atributů

$$T = (\texttt{TABLE\_CATALOG}, \texttt{TABLE\_SCHEMA}, \texttt{TABLE\_NAME})$$

COLUMNS

Řádek je jednoznačně identifikován čtveřicí atributů

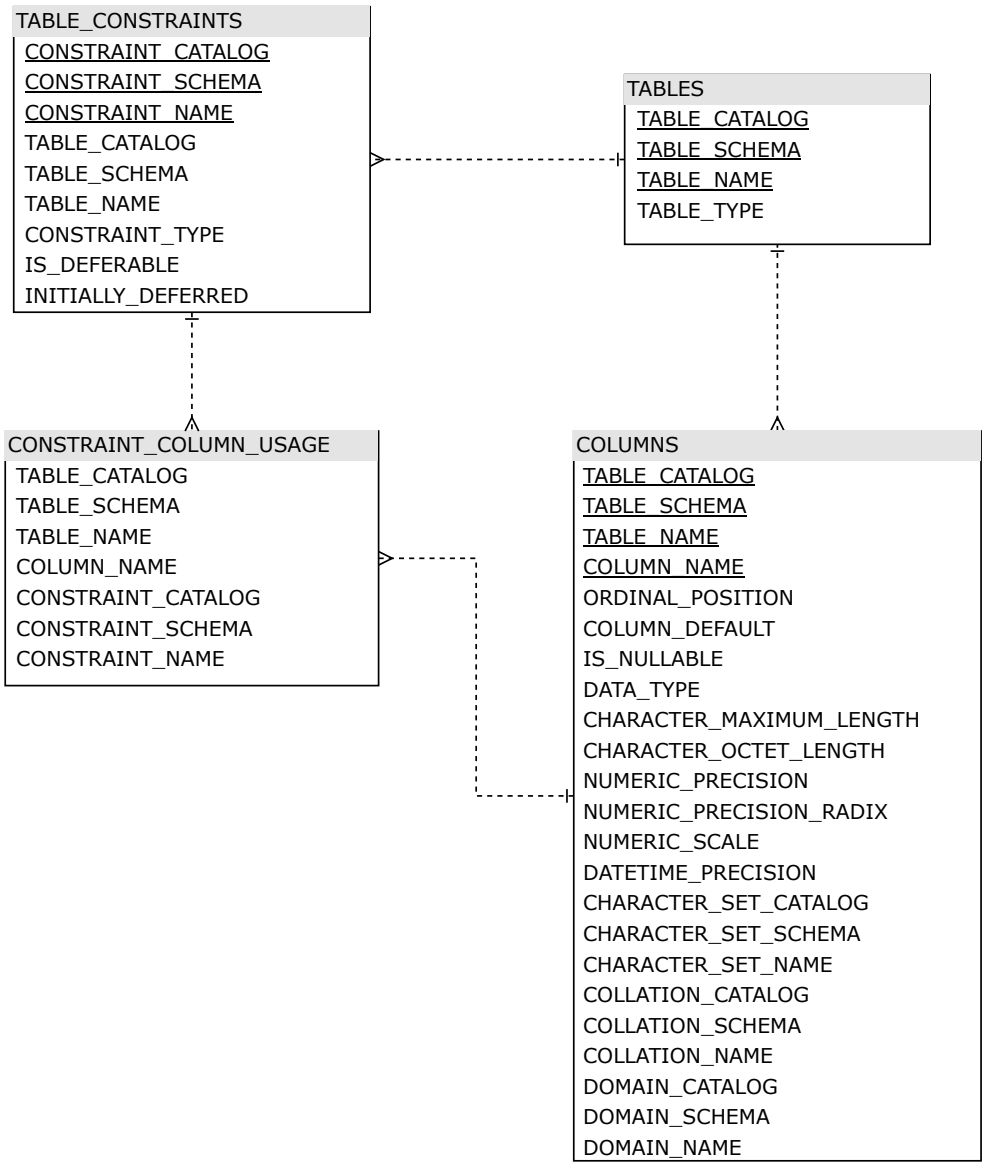
$$C = (\texttt{TABLE\_CATALOG}, \texttt{TABLE\_SCHEMA}, \texttt{TABLE\_NAME}, \texttt{COLUMN\_NAME})$$

TABLE_CONSTRAINTS

Řádek je jednoznačně identifikován trojicí

$$TC = (\texttt{CONSTRAINT\_CATALOG}, \texttt{CONSTRAINT\_SCHEMA}, \texttt{CONSTRAINT\_NAME})$$

. Je možné vytvořit vazbu přes T na `TABLES`.



Obrázek 5.2: Struktura INFORMATION_SCHEMA

```
SELECT
  T.CONSTRAINT_CATALOG,
  T.CONSTRAINT_SCHEMA,
  T.CONSTRAINT_NAME,
  T.CONSTRAINT_TYPE,
  T.TABLE_CATALOG,
  T.TABLE_SCHEMA,
  T.TABLE_NAME,
  C.COLUMN_NAME
FROM
  INFORMATION_SCHEMA.TABLE_CONSTRAINTS T
JOIN
  INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE C
ON
  C.CONSTRAINT_CATALOG = T.CONSTRAINT_CATALOG AND
  C.CONSTRAINT_SCHEMA = T.CONSTRAINT_SCHEMA AND
  C.CONSTRAINT_NAME = T.CONSTRAINT_NAME AND
  C.TABLE_NAME = T.TABLE_NAME
WHERE
  T.CONSTRAINT_TYPE IN ('PRIMARY KEY', 'FOREIGN KEY', 'UNIQUE')
```

Ukázka kódu 5: Dotaz pro získání informací o Primárních klíčích a Unique omezeních

CONSTRAINT_COLUMN_USAGE

Vazební entita, která propojuje `TABLE_CONSTRAINTS` přes T a `COLUMNS` přes C.

5.2.3 Načtení integritních omezení

Integritní omezení dostupná v `TABLE_CONSTRAINTS` jsou typů:

- **PRIMARY KEY** – Primární klíč.
- **FOREIGN KEY** – Cizí klíč.
- **UNIQUE** – Unikátní hodnoty ve sloupci.
- **CHECK** – Definovaný požadavek, který musí být splněn.

Integritní omezení typu primární klíč, cizí klíč a UNIQUE je možné získat například dotazem uvedeným v ukázce kódu 5

```
SELECT
  SRC.CONSTRAINT_CATALOG AS 'SRC_CONSTRAINT_CATALOG',
  SRC.CONSTRAINT_SCHEMA AS 'SRC_CONSTRAINT_SCHEMA',
  SRC.CONSTRAINT_NAME AS 'SRC_CONSTRAINT_NAME',
  SRC.TABLE_CATALOG AS 'SRC_TABLE_CATALOG',
  SRC.TABLE_SCHEMA AS 'SRC_TABLE_SCHEMA',
  SRC.TABLE_NAME AS 'SRC_TABLE_NAME',
  SRC.COLUMN_NAME AS 'SRC_COLUMN_NAME',
  DST.CONSTRAINT_CATALOG AS 'DST_CONSTRAINT_CATALOG',
  DST.CONSTRAINT_SCHEMA AS 'DST_CONSTRAINT_SCHEMA',
  DST.CONSTRAINT_NAME AS 'DST_CONSTRAINT_NAME',
  DST.TABLE_CATALOG AS 'DST_TABLE_CATALOG',
  DST.TABLE_SCHEMA AS 'DST_TABLE_SCHEMA',
  DST.TABLE_NAME AS 'DST_TABLE_NAME',
  DST.COLUMN_NAME AS 'DST_COLUMN_NAME'
FROM INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS RC
JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE SRC ON
  SRC.CONSTRAINT_CATALOG = RC.CONSTRAINT_CATALOG AND
  SRC.CONSTRAINT_SCHEMA = RC.CONSTRAINT_SCHEMA AND
  SRC.CONSTRAINT_NAME = RC.CONSTRAINT_NAME
JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE DST ON
  DST.CONSTRAINT_CATALOG = RC.UNIQUE_CONSTRAINT_CATALOG AND
  DST.CONSTRAINT_SCHEMA = RC.UNIQUE_CONSTRAINT_SCHEMA AND
  DST.CONSTRAINT_NAME = RC.UNIQUE_CONSTRAINT_NAME
WHERE SRC.ORDINAL_POSITION = DST.ORDINAL_POSITION
```

Ukázka kódu 6: Dotaz pro získání informací o cizích klíčích. [25]

Cizí klíče

Pokud bychom chtěli implementovat algoritmus mapování schémat založený na struktuře schémat a vazbách mezi tabulkami, bylo by nutné získat u cizích klíčů i informaci o tom, jakou tabulku klíč referencuje. K tomu je možné využít pohledy `REFERENTIAL_CONSTRAINTS` a `KEY_COLUMN_USAGE`. Navržený dotaz je přiložen v ukázce kódu 5.2.3.

5.3 Využité externí knihovny

- **Math.NET** – Maticové a další matematické operace.
- **Accord.NET** – Implementace Munkresova algoritmu.
- **F23.StringSimilarity** – Podobnostní funkce textových řetězců.

- **BlazorFluentUI** – Komponenty pro vývoj uživatelského rozhraní.

5.4 Kompatibilita datových typů MSSQL

Jedním z faktorů, který určuje, jestli jsou si dva databázové sloupce podobné je obtížnost konverze zdrojového datového typu na cílový. Kompatibilita datových typů v MSSQL je znázorněna na obrázku 5.3. Rozlišeno je několik možných typů konverze:

- **explicitní konverze** – uživatel musí konverzi explicitně definovat pomocí SQL funkcí `CAST`, nebo `CONVERT`
- **implicitní konverze** – je provedena automaticky bez zásahu uživatele
- **konverze není dovolena** – není možné provést konverzi přímo mezi těmito typy, je ale možné použít více konverzí
- **konverze vyžaduje explicitní `CAST`** kvůli možné ztrátě přesnosti

5.5 Nalezení mapování

V sekci 3.5.3 byl popsán Kuhn-Munkersův algoritmus pro nalezení optimálního přiřazení. Tento algoritmus implementuje v .NET například knihovna Accord.NET a její třída `Munkres`.[26] V ukázce kódu 7 je tato třída využita pro nalezení maximálního přiřazení na podobnostní matici M o rozměrech 3x3.

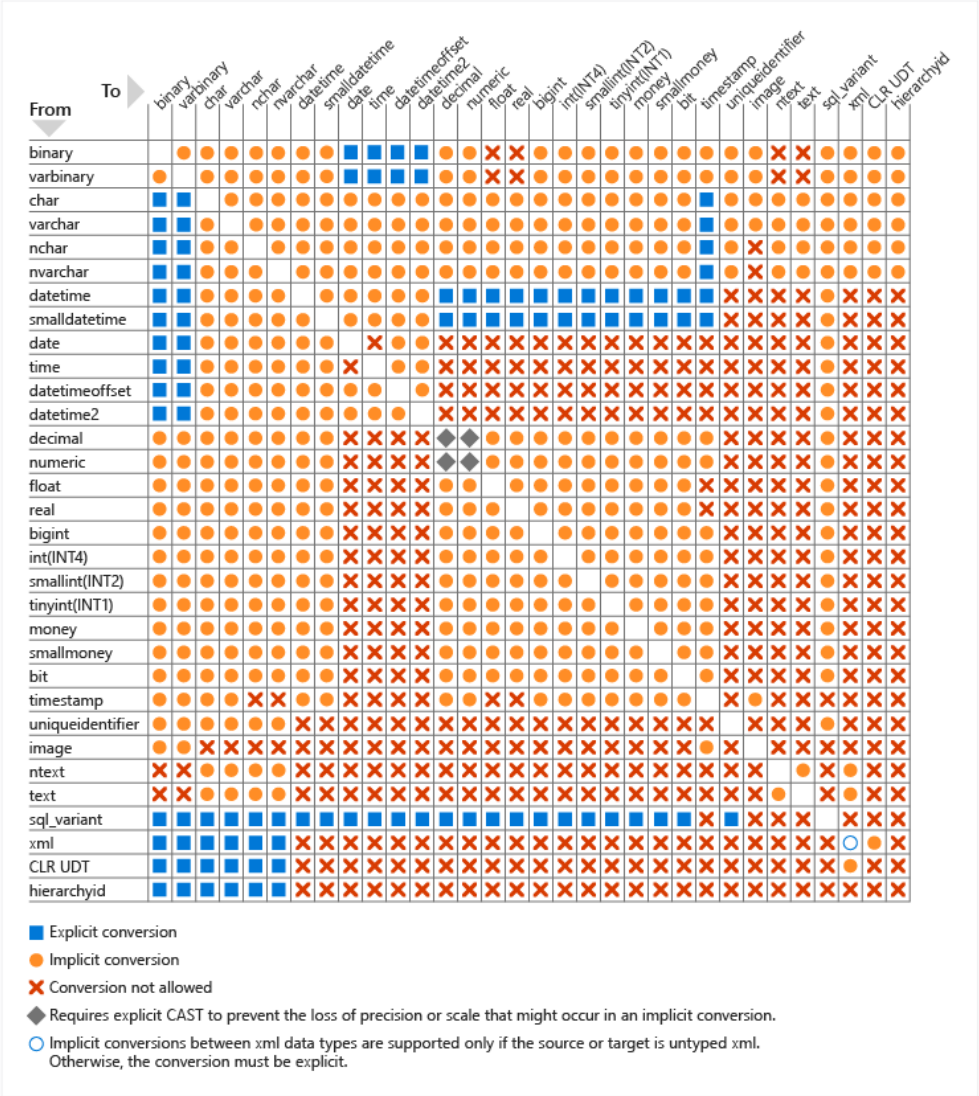
$$M = \begin{pmatrix} \mathbf{4} & 1 & 3 \\ 3 & 2 & \mathbf{3} \\ 1 & \mathbf{5} & 4 \end{pmatrix}$$

Do řádků můžeme dosadit např. atributy zdrojového schématu a do sloupců atributy cílového schématu. Hodnoty matice pak reprezentují jejich podobnost. Řešení $0, 1, 2$ každému řádku přiřazuje index cílového sloupce, indexováno od 0. Hodnota tohoto přiřazení je pak:

$$M_{0,0} + M_{1,2} + M_{2,1} = 4 + 3 + 5 = 12$$

5.6 Webová aplikace

Aplikace je vytvořena pomocí Razor komponent. Komponenta je samostatná část uživatelského rozhraní, která umožňuje dynamické chování. Komponenty mohou být vnořené, opakovaně používané a sdílené mezi projekty. [27]



Obrázek 5.3: Tabulka kompatibility datových typů v MSSQL

```
var similarityMatrix = new double[][]
{
    new double[] { 4, 1, 3 },
    new double[] { 3, 2, 3 },
    new double[] { 1, 5, 4 }
};

var munkres = new Munkres(similarityMatrix);
munkres.Maximize();

double[] solution = munkres.Solution;
_output.WriteLine("Solution: " + string.Join(",", solution));

//Standard Output:
//Solution: 0,2,1
```

Ukázka kódu 7: Použití třídy Munkres v knihovně Accord.NET

Jednoduchým příkladem je komponenta `DeleteConfirmationModal`. Je využívána při zobrazení potvrzovacího okna po kliknutí na smazání projektu, nebo správce připojení. Umožňuje pomocí parametrů zadat text, který se má zobrazit a nastavit `EventCallback`, který má komponenta zavolat v rodiči po kliknutí na některé její tlačítko.

Snímky implementované webové aplikace jsou dostupné v příloze C.

Kapitola 6

Testování

6.1 Jednotkové testy

Pro vytvoření jednotkových (Unit) testů byla použita knihovna xUnit[9]. Testy byly implementovány v rámci projektu `AutoMapper.Test`. Jejich cílem je ověřit správné fungování objektů a jejich metod. Správné výstupy jsou ověřovány za pomoci knihovny FluentAssertions[10].

Pokud jsou testy implementovány v jiném projektu, než testované třídy, je standardně možné testovat pouze třídy s viditelností `public`. Aby bylo možné testovat i třídy s viditelností `internal`, tedy třídy, které jsou přístupné pouze ze stejné assembly, Je potřebné přidat do příslušných souborů .csproj deklaraci `<InternalsVisibleTo Include="AutoMapper.Test" />`

6.2 Uživatelské testy

Uživatelské testování bylo provedeno s vývojáři datových integrací. Testeři byli požádáni, aby před testováním dodali vybrané zdrojové a cílové tabulky, mezi kterými chtějí nalézt mapování. Dále byli požádáni, aby se struktura těchto tabulek nějakým způsobem lišila, tak aby bylo možné ověřit funkcionalitu automatického mapování.

Na základě poskytnutých schémat byly testerům v testovací databázi vytvořena zdrojová a cílová schémata a v nich příslušné tabulky.

Před zahájením samotného testování byli testeři krátce seznámeni s aplikací. Poté dostali k dispozici seznam úkolů, které mají postupně splnit. Tento seznam je stejný, jako seznam využitý při testování samotného uživatelského rozhraní, popsaný v sekci 4.1.3. Dále dostali k dispozici údaje pro připojení do testovací databáze a jména schémat, ve kterých se jejich tabulky nacházejí.

[9] `https://xunit.net/`
[10] `https://fluentassertions.com/`

6.2.1 Shrnutí uživatelských testů

Připomínky testerů k návrhu, nebo funkcionalitě aplikace jsou shrnuty v následujících bodech, které odpovídají krokům testovacího scénáře. V závorce u každé připomínky je uvedeno, jestli již úprava byla implementována. Úpravy, které ještě nebyly implementovány byly zařazeny do seznamu úkolů k vyřešení.

2. Vyplnit parametry správce připojení

- Pokud uživatel vyplní připojovací řetězec do databáze, měly by se ostatní položky formuláře deaktivovat, aby uživatel neměl tendenci je také vyplnit. (Ne)

3. Otestovat, jestli je připojení funkční

- Po stisknutí tlačítka `Test connection` by se měl zobrazit text, který indikuje úspěch, nebo chybu. Nyní se pouze změní barva tlačítka na zelenou, nebo červenou. (Ano)

5. Vyplnit parametry projektu

- Typy entit (tabulka, pohled) by měly být předvybrány. (Ano)
- Seznamy schémat, typů entit a tabulek by měly obsahovat možnost hromadně vybrat, nebo odebrat všechny položky v seznamu. (Ne)
- Jednotlivé položky formuláře by se měly postupně aktivovat podle toho, jak uživatel vyplňuje. (Ne)

7. Spustit mapování

- Pokud je projekt vytvořen, měl by návrh mapování proběhnout automaticky, nyní vyžaduje kliknutí uživatelem na tlačítko `Generate mapping` (Ne)
- U entit, jejichž jména se liší, ale atributy jsou si velmi podobné má jméno příliš velkou váhu a tím snižuje celkovou podobnost. Řešením může být zavedení vážení podle počtu podobných atributů. (Ne)

9. Potvrdit mapování všech entit, která mají podobnost větší, než 90 %

- Chybí možnost hromadného potvrzení, mapování je potřeba potvrdit, nebo zamítnout manuálně. (Ne)
- Detail entity by mělo být možné otevřít, ikdyž už je mapování potvrzeno, například v režimu pouze pro čtení. (Ne)

- Podobnost by měla být podpořena barevně, aby bylo možné snadno vizuálně rozlišit podobné a odlišné entity. (Ne)

10. Otevřít detail entity

- Položky, které obsahují velmi dlouhé názvy sloupců přetékají, mohlo by být vyřešeno zvětšením modalového okna. (Ano)

11. Upravit mapování atributů

- Při výběru zdrojového atributu by měly být atributy:
  - Seřazeny podle podobnosti.
  - Odlišeny ty, které již jsou namapovány na jiný cílový atribut.
  - Při přiřazení atributu uživatelem by mělo dojít k automatickému přepočítání podobností, které již bude pracovat pouze ze zbývajícími atributy. (Ne)

13. Uložit změny v mapování

- Chybí zobrazení potvrzení, že byly změny uloženy. (Ne)

Závěr

V rámci této diplomové práce byla nejprve provedena rešerše metod, které mohou být použity pro porovnávání a nalezení mapování mezi databázovými schématy. Byly představeny metody porovnávající názvy objektů některou podobnostní metrikou, metody využívající další metadata atributů a metody založené na strojovém učení.

Dalším úkolem bylo provést rešerši nástrojů používaných v odvětví datových integrací. V rešerši bylo ukázáno, že tyto nástroje umí navrhnout mapování mezi atributy pouze při úplné shodě názvů. Mapování mezi entitami nezvládnou detekovat vůbec. Vytváření mapování mezi entitami je repetetivní a časově náročná činnost, kterou musí vývojář provést manuálně.

Na základě provedených rešerší byla navržena a implementována webová aplikace, která na vstupu dostane zdrojové a cílové databázové schéma a automaticky vytvoří mapování na základě podobnosti s využitím popsaných metod. Uživatel aplikace může případně navržené mapování upravit a exportovat jej pro využití v některém integračním nástroji.

Poslední část práce tvoří testování. Nejprve proběhlo testování nad vytvořeným prototypem, během kterého bylo postupně vylepšováno uživatelské rozhraní. Poté byla otestována také vytvořená aplikace. Na základě testování byla navržena vylepšení, která budou zpracována v dalších verzích.

Testování s uživateli, kteří aktuálně vyvíjejí integrační řešení ukázalo, že aplikace má v tomto odvětví potenciál. Největšího využití dosáhne u větších projektů, ve kterých je nutné integrovat velké množství entit a práce v tradičních ETL nástrojích je tak velmi neefektivní.

Výstup implementované aplikace, tedy vygenerované mapování ve formátu JSON, je možné využít například v některé ze zmíněných knihoven zaměřených na integrační procesy.

Další rozvoj

Prozatím je aplikace omezena pouze na Microsoft SQL Server. Pro plnohodnotné využití a rozšíření je za potřebí implementovat připojení k dalším zdrojům. Mezi ty mohou patřit další databázové systémy, jako např. Oracle, nebo PostgreSQL, ale také REST API za využití specifikace Open API, nebo soubory typů csv, xslx a json.

Dalším možným vylepšením je implementace dalších mapovacích algoritmů, které umožní překonat další konflikty, popsané v sekci 1.5.1, které mohou nastat mezi schématy. Příkladem je problém při mapování 1:N, nebo dokonce M:N.

Jedním z úkolů, který řeší vývojáři integračních procesů je také definování závislostí mezi integracemi tak, aby nebyla porušena referenční integrita příslušných databázových systémů. Tento proces by bylo také možné automatizovat na základě získaných metadat o cizích klíčích, z nichž je možné vytvořit graf závislostí, podle kterého může integrační proces postupuvat.

Aby bylo možné efektivně vyhodnotit kvalitu navržených mapování, je za potřebí vytvořit sadu mapovacích úkolů a manuálně je vyhodnotit. Poté bude možné určit metriky, jako jsou Precision a Recall.

Vhodným rozšířením může být také použití strojového učení pro zpřesnění výsledku. Může se jednat o supervizované metody natrénované na předem připravených datech, nebo o nesupervizované metody využívající například klastrování.

Literatura

[1] Kimball, R.; Ross, M.: *The data warehouse toolkit*. Indianapolis: Wiley, třetí vydání, c2013, ISBN 978-111-8530-801.

[2] Madhavan, J.; Bernstein, P. A.; Doan, A.; aj.: Corpus-Based Schema Matching. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, USA: IEEE Computer Society, 2005, ISBN 0769522858, str. 57–68, doi:10.1109/ICDE.2005.39. Dostupné z: https://doi.org/10.1109/ICDE.2005.39

[3] Navarro, G.: A Guided Tour to Approximate String Matching. *ACM Comput. Surv.*, ročník 33, č. 1, mar 2001: str. 31–88, ISSN 0360-0300, doi:10.1145/375360.375365. Dostupné z: https://doi.org/10.1145/375360.375365

[4] Friendly, F.: Jaro–Winkler Distance Improvement For Approximate String Search Using Indexing Data For Multiuser Application. *Journal of Physics: Conference Series*, ročník 1361, č. 1, nov 2019: str. 012080, doi:10.1088/1742-6596/1361/1/012080. Dostupné z: https://doi.org/10.1088/1742-6596/1361/1/012080

[5] Sahay, T.; Mehta, A.; Jadon, S.: Schema Matching using Machine Learning. In *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2020, s. 359–366, doi:10.1109/SPIN48934.2020.9071272.

[6] Kim, W.; Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, ročník 24, č. 12, 1991: s. 12–18, doi:10.1109/2.116884.

[7] Celko, J.; Books24x7, I.: *Joe Celko's SQL programming style*. Amsterdam: Elsevier, 2005, ISBN 0120887975;9780120887972;.

[8] Oracle E-Business Suite Developer's Guide [online]. [2022-02-16]. Dostupné z: https://docs.oracle.com/cd/E18727_01/doc.121/e12897/T302934T458266.htm

[9] DeepL API [online]. [2022-02-16]. Dostupné z: https://www.deepl.com/docs-api/translating-text/request/

[10] Do, H.-H.; Rahm, E.: COMA—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, Elsevier, 2002, s. 610–621.

[11] Apache Airflow [online]. 2022, [cit. 2022-01-20]. Dostupné z: https://airflow.apache.org/

[12] Github - spotify/luigi [online]. 2021, [cit. 2022-01-20]. Dostupné z: https://github.com/spotify/luigi

[13] Openflights: Airports and airline data [online]. [cit. 2022-01-31]. Dostupné z: https://openflights.org/data.html

[14] SQL Server Integration Services (SSIS). 2022, [cit. 2022-01-27]. Dostupné z: https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-ver15

[15] Microsoft: Data Flow - SQL Server Integration Services (SSIS) [online]. [cit. 2022-04-23]. Dostupné z: https://docs.microsoft.com/en-us/sql/integration-services/data-flow/data-flow?view=sql-server-ver15

[16] Bouman, R.; Casters, M.; van Dongen, J.: *Pentaho Kettle Solutions*. Indianapolis: John Wiley & Sons, Incorporated, první vydání, 2010, ISBN 978-0-470-63517-9.

[17] Azure Data Factory [online]. [cit. 2022-01-31]. Dostupné z: https://docs.microsoft.com/en-us/azure/data-factory/introduction

[18] Azure Data Factory & Azure Synapse [online]. [cit. 2022-01-31]. Dostupné z: https://docs.microsoft.com/en-us/azure/data-factory/copy-activity-schema-and-type-mapping

[19] Relational databases - ETLBox [online]. [cit. 2022-04-29]. Dostupné z: https://www.etlbox.net/docs/db-connectors/relational-databases

[20] Kuhn, H. W.: The Hungarian method for the assignment problem. *Naval research logistics*, ročník 52, č. 1, 2005: s. 7–21.

[21] Nielsen, J.: Enhancing the Explanatory Power of Usability Heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, New York, NY, USA: Association for Computing Machinery, 1994, ISBN 0897916506, str. 152–158, doi:10.1145/191666.191729. Dostupné z: https://doi.org/10.1145/191666.191729

[22] ASP.NET Core Blazor hosting models [online]. [2022-04-26]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0

[23] Creating and configuring a mode [online]l. [2022-05-01]. Dostupné z: https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database

[24] System Information Schema Views (Transact-SQL) [online]. [2022-05-01]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/system-information-schema-views/system-information-schema-views-transact-sql?view=sql-server-ver15

[25] SQL Essentials: Foreign Key Metadata: Improving on INFORMATION_SCHEMA Views [online]. [2022-03-22]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/sql/legacy/aa175805(v=sql.80)?redirectedfrom=MSDN

[26] Munkres Class [online]. [2022-04-06]. Dostupné z: http://accord-framework.net/docs/html/T_Accord_Math_Optimization_Munkres.htm

[27] ASP.NET Core Razor components [online]. [2022-05-01]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-6.0

Příloha **A**

Seznam použitých zkratek

ETL Extract Transform Load

CRM Customer Relationship Management

ERP Enterprise Resource Planning

SCM Supply Chain Management

ODBC Open Database Connectivity

OLE DB Object Linking and Embedding, Database

SQL Structured Query Language

SSIS SQL Server Integration Services

JSON Javascript Object Notation

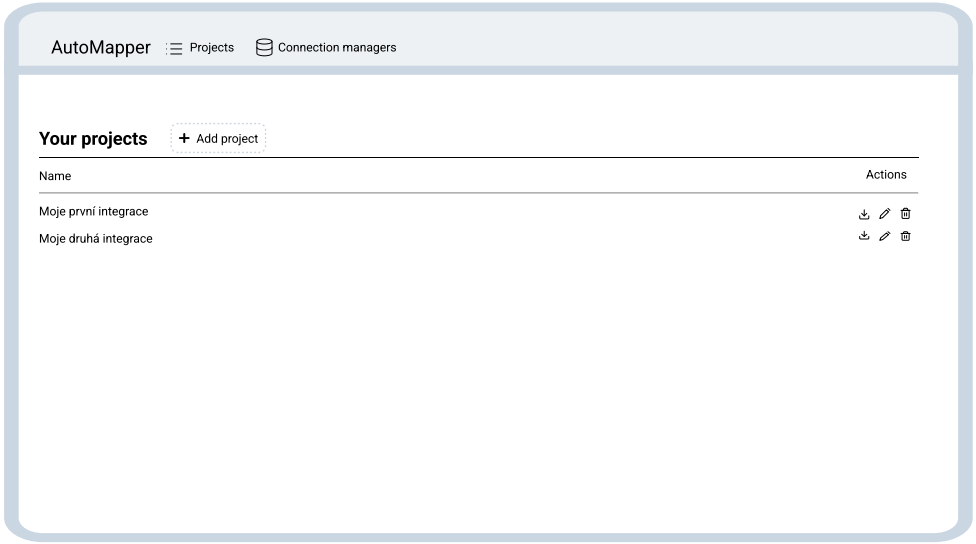
API Application Programming Interface

PK Primary Key/Primární klíč

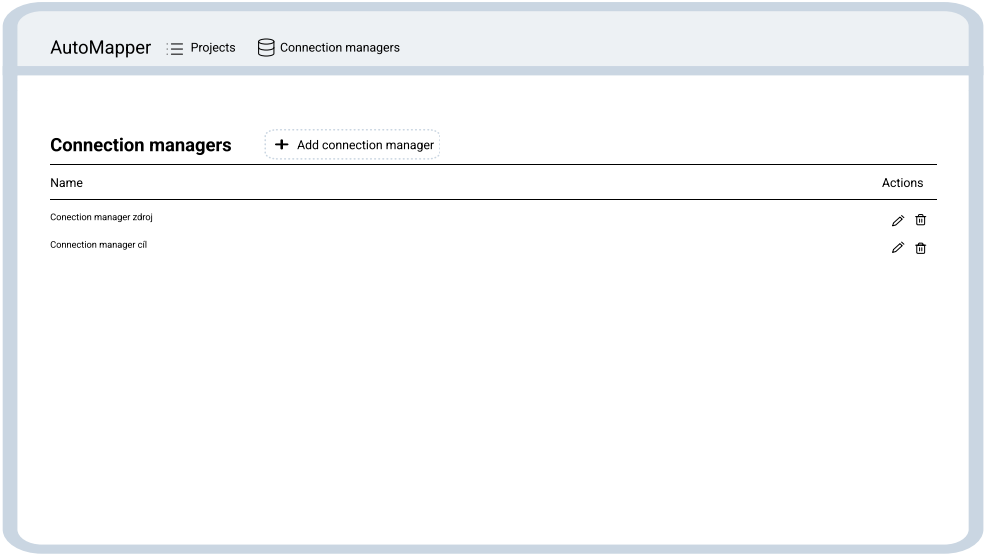
FK Foreign Key/Cizí klíč

Příloha B

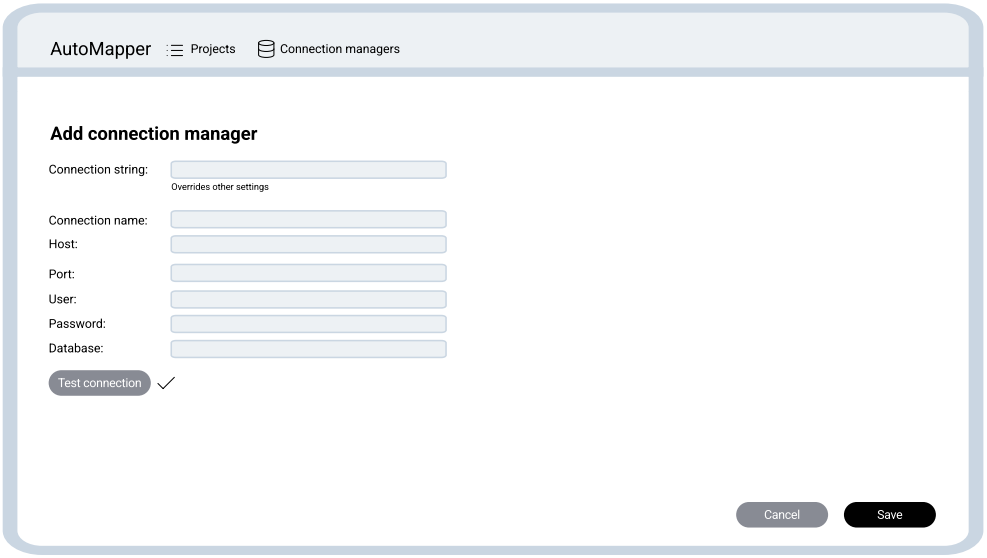
Wireframe prototyp



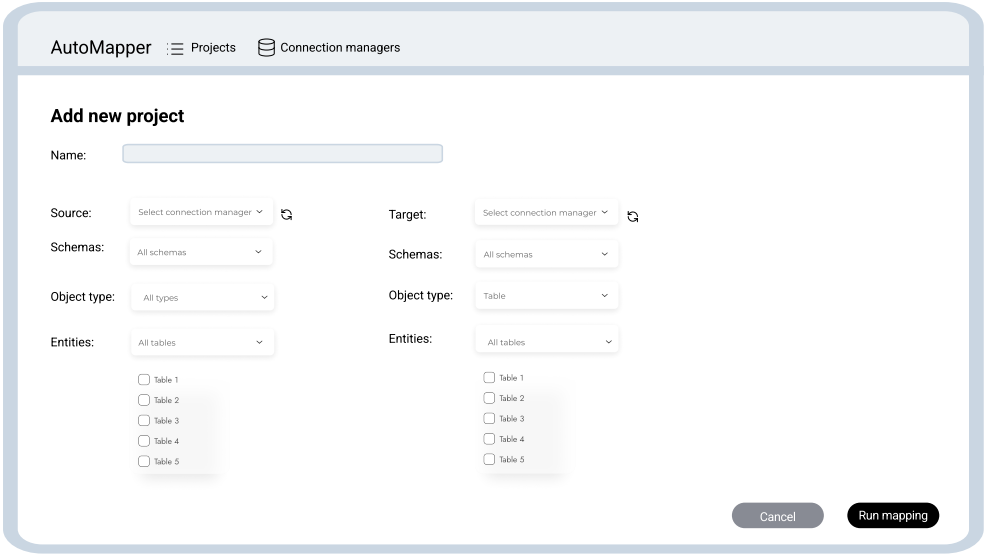
Obrázek B.1: Prototyp – Přehled projektů



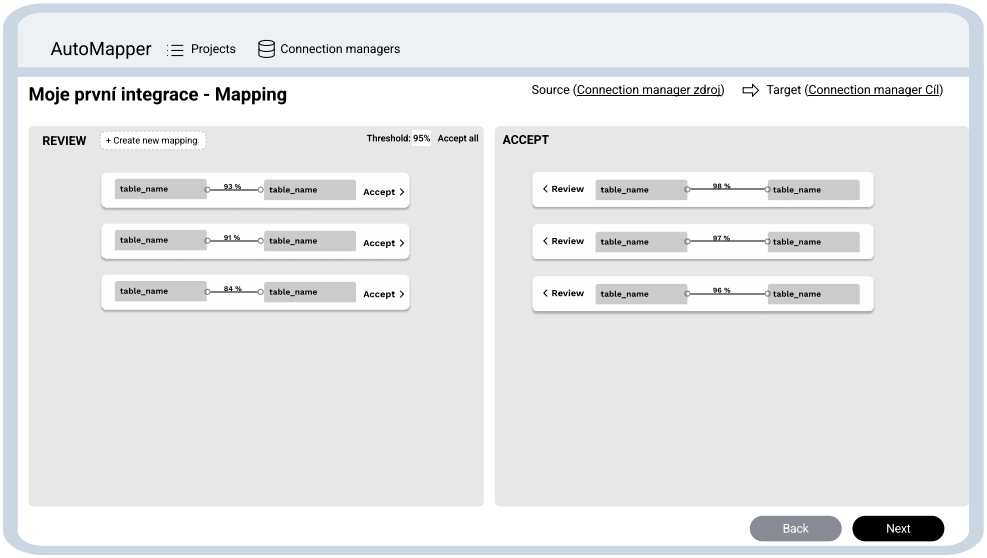
Obrázek B.2: Prototyp – Přehled správců připojení



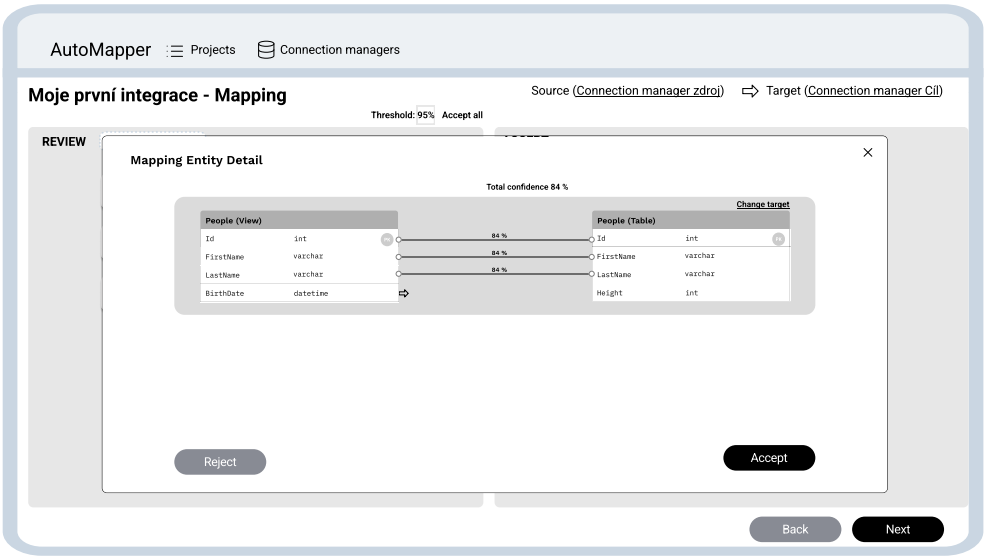
Obrázek B.3: Prototyp – Vytvoření správce připojení



Obrázek B.4: Prototyp – Vytvoření projektu



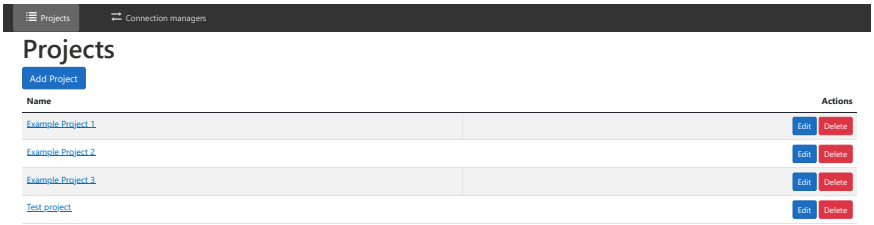
Obrázek B.5: Prototyp – Detail projektu



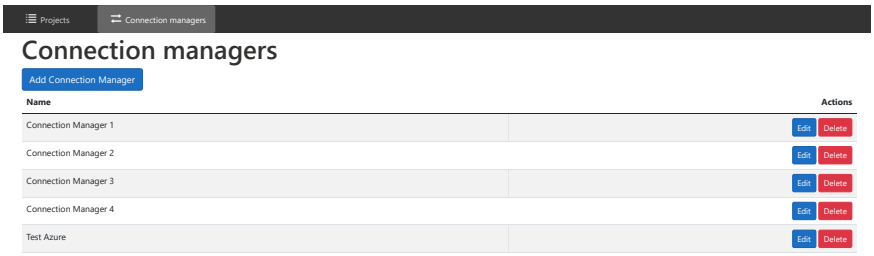
Obrázek B.6: Detail mapované entity

Příloha C

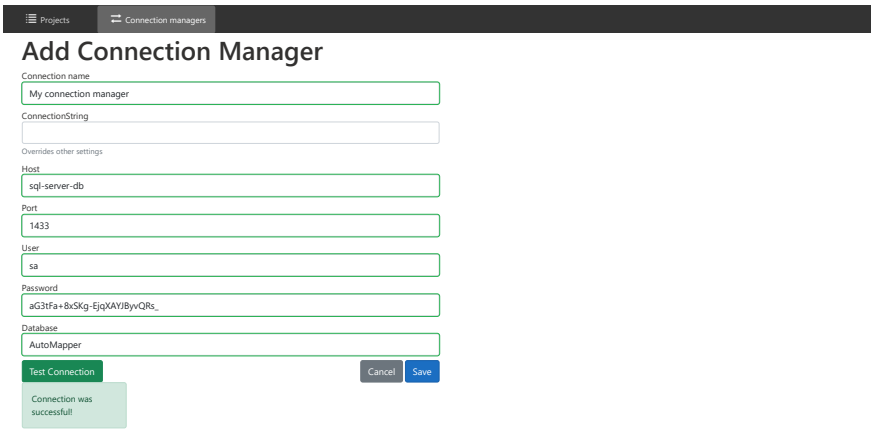
Snímky aplikace



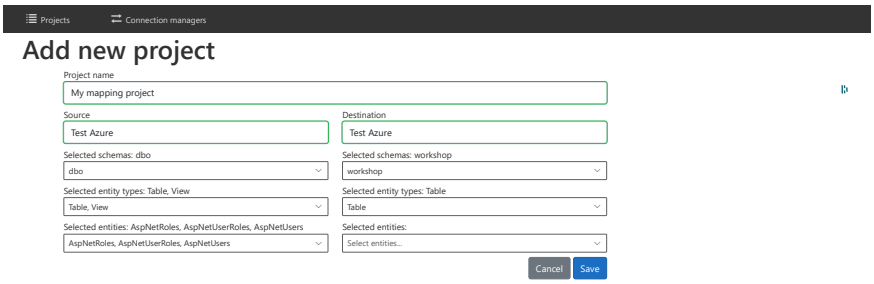
Obrázek C.1: Přehled projektů



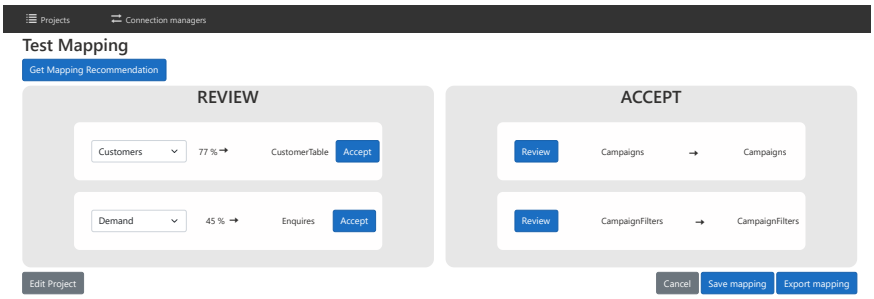
Obrázek C.2: Přehled správců připojení



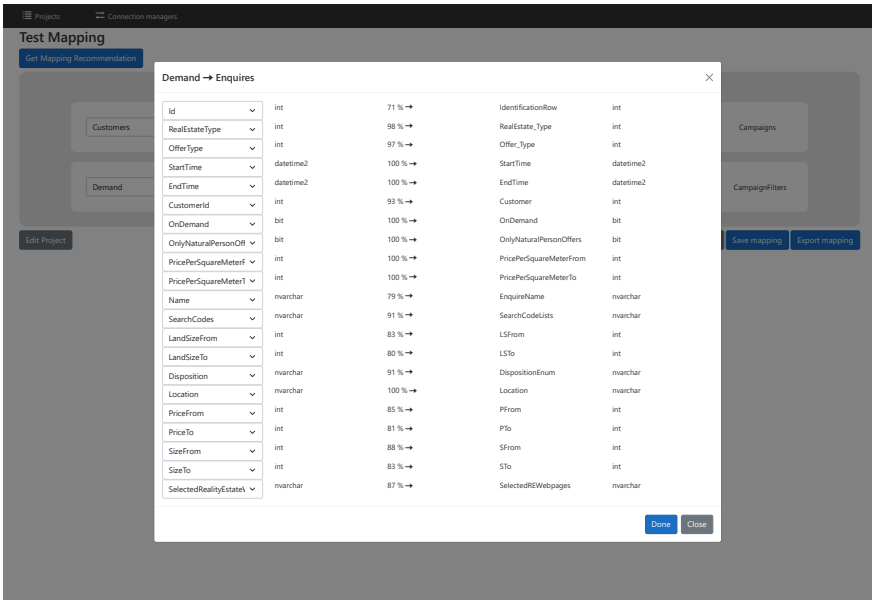
Obrázek C.3: Vytvoření správce připojení



Obrázek C.4: Vytvoření projektu



Obrázek C.5: Vytvoření správce připojení



Obrázek C.6: Vytvoření správce připojení

Příloha D

Nasazení aplikace

D.1 Docker compose

Preferovaným způsobem, jak aplikaci nasadit je využití nástroje Docker Compose. V kořenovém adresáři řešení je vytvořen soubor `docker-compose.yml`, který definuje potřebné kontejnery a jejich závislosti. Tento soubor je přiložen jako ukázka kódu 8. Pro správný běh aplikace jsou potřebné následující tři služby.

Microsoft SQL Server – sql-server-db

Jde o oficiální Microsoft SQL Server Docker image, který je využíván s licencí Developer.[11] Sql server je dostupný na standardně používaném portu 1433. Používá Docker Volume pojmenovaný `sqlvolume`, aby uložená data v kontejneru byla perzistována.

Backendové ASP.NET Core API – automapper/api

Tato služba spouští implementované ASP.NET Core API. Kontejner využívá image .NET SDK. [12] API je dostupné na portu 7055. Při spuštení kontejneru je nastavena proměnná prostředí `ASPNETCORE_ENVIRONMENT=Docker`, aby došlo k načtení konfigurace specifické pro běh v Dockeru.

Blazor webová aplikace – web

Webová aplikace vychází stejně jako API z image .NET SDK. Je dostupná na adrese `localhost:80`.

Výše popsané služby je možné sestavit a spustit příkazem:

```
docker-compose up
```

[11] `https://hub.docker.com/_/microsoft-mssql-server`
[12] `https://hub.docker.com/_/microsoft-dotnet-sdk`

Správné spuštění je možné ověřit příkazem `docker ps`. Spuštěny by měly být tři kontejnery: `mcr.microsoft.com/mssql/server`, `automapper/api` a `automapper/web`.

D.2 Spuštění pomocí .NET SDK

Aplikaci je možné spustit také pomocí .NET 6.0 SDK[13]. Tento způsob je využívaný hlavně při vývoji.

Databáze

Jako vývojový databázový systém je použita `LocalDB`. Jde o minimální verzi SQL serveru, která je určena pro lokální vývojové účely. Alternativně může být vytvořena databáze například v cloudu.

Inicializace databáze – vytvoření datových struktur a naplnění počátečními daty je provedeno automaticky při startu backendové aplikace.

Backendové ASP.NET Core API

Projekt je umístěn v adresáři AutoMapper.Backend. Sestavit a spustit jej je možné příkazem `dotnet run`. API je pak dostupné na adrese `localhost:7055/`. Swagger dokumentace k API endpointům a metodám je dostupná na adrese `localhost:7055/swagger/index.html`.

Blazor webová aplikace

Projekt je umístěn v adresáři AutoMapper.Backend. Sestavit a spustit jej je možné příkazem `dotnet run`. Webová aplikace je dostupná na adrese `localhost:7099/`

D.3 Struktura přiloženého média

Obrázek D.1: Struktura přiloženého média

- `readme.txt` ..... Popis obsahu média
- `src` ..... Adresář se zdrojovými kódy
  - `application` ..... Zdrojové kódy aplikace
    - `readme.txt` ..... Manuál pro spuštení aplikace
  - `text` ..... Zdrojové kódy textu
- `thesis.pdf` ..... Tato diplomová práce ve formátu PDF

[13] https://dotnet.microsoft.com/en-us/download/dotnet/6.0

```yaml
version: "3.9"
services:
  sql-server-db:
    image: mcr.microsoft.com/mssql/server
    ports:
      - "1433:1433"
    environment:
      SA_PASSWORD: "my_password"
      ACCEPT_EULA: "Y"
    volumes:
      - sqlvolume:/var/opt/mssql
  api:
    image: automapper/api
    build:
        context: .
        dockerfile: ./AutoMapper.Backend/Dockerfile
    ports:
        - "7055:80"
    depends_on:
        - sql-server-db
    environment:
        - ASPNETCORE_ENVIRONMENT=Docker
  web:
    image: automapper/web
    build:
        context: .
        dockerfile: ./AutoMapper.Web/Dockerfile
    ports:
        - "80:80"
    depends_on:
        - api
volumes:
  sqlvolume:
```

Ukázka kódu 8: Soubor docker-compose.yml