**MORAVSKÁ VYSOKÁ ŠKOLA OLOMOUC**

Ústav informatiky a aplikované matematiky

**Re-Engineering nástroje DBSC a procesu generování databáze**

BAKALÁŘSKÁ PRÁCE

Jaroslav Sýkora

Vedoucí práce: Ing. Jiří Blahuta, Ph.D.

Olomouc 2021

**Prohlášení**

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a použil jen zdroje v seznamu literatury a použitých zdrojů.

Tištěná verze textu práce je shodná s textem práce na CD nosiči a elektronickou verzí vloženou do studijního systému IS/STAG.

V Olomouci ………………………. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Jaroslav Sýkora

**Poděkování**

Touto cestou bych rád poděkoval svému vedoucímu práce Ing. Jiřímu Blahutovi, Ph.D. za cenné rady, ochotu a trpělivost v průběhu zpracování této bakalářské práce.

**Obsah**

[Úvod 6](#_Toc70622767)

[I. Teoretická část 7](#_Toc70622768)

[1 Představení původní aplikace 7](#_Toc70622769)

[1.1 Popis fungování aplikace 7](#_Toc70622770)

[2 Zadání 9](#_Toc70622771)

[3 Jazyk C# 9](#_Toc70622772)

[3.1 Základní prvky jazyka C# 11](#_Toc70622773)

[4 Extensible Markup Language 12](#_Toc70622774)

[4.1 Struktura XML dokumentu 12](#_Toc70622775)

[4.2 Document Type Definition 13](#_Toc70622776)

[4.3 Zpracování XML dokumentu 14](#_Toc70622777)

[5 Reverzní inženýrství 15](#_Toc70622778)

[6 Volba frameworku pro uživatelské rozhraní 15](#_Toc70622779)

[6.1 Blazor 15](#_Toc70622780)

[6.1.1 Rozdíl mezi Razor Pages a Blazorem 15](#_Toc70622781)

[6.2 Electron 18](#_Toc70622782)

[6.3 Vyhodnocení 19](#_Toc70622783)

[7 Databáze 19](#_Toc70622784)

[7.1 Typy databází 19](#_Toc70622785)

[7.2 Structured Query Language 20](#_Toc70622786)

[7.3 Systém řízení báze dat (SŘBD) 21](#_Toc70622787)

[7.4 Datový model 21](#_Toc70622788)

[II. Praktická část 23](#_Toc70622789)

[8 Reengineering původní aplikace 23](#_Toc70622790)

[8.1 Uživatelský vstup 23](#_Toc70622791)

[8.2 Proces generování databáze 24](#_Toc70622792)

[9 Struktura řešení 24](#_Toc70622793)

[9.1 User Interface 24](#_Toc70622794)

[9.1.1 Client 25](#_Toc70622795)

[9.1.1 Server 27](#_Toc70622796)

[9.2 Business 28](#_Toc70622797)

[9.2.1 Comparer 29](#_Toc70622798)

[9.2.2 DiffMessages 29](#_Toc70622799)

[9.2.3 Parsers 30](#_Toc70622800)

[9.2.4 Dbsc 31](#_Toc70622801)

[9.2.5 ScriptComposer 32](#_Toc70622802)

[9.3 Tests 35](#_Toc70622803)

[10 Finální aplikace 36](#_Toc70622804)

[11 Publikování aplikace 39](#_Toc70622805)

[11.1 Publikování aplikace do složky 39](#_Toc70622806)

[11.2 Obfuskace aplikace 43](#_Toc70622807)

[Závěr 44](#_Toc70622808)

[POUŽITÁ LITERATURA 45](#_Toc70622809)

[OSTATNÍ ZDROJE 46](#_Toc70622810)

[SEZNAM ZKRATEK 49](#_Toc70622811)

[SEZNAM OBRÁZKŮ 50](#_Toc70622812)

[SEZNAM ZDROJOVÝCH KÓDŮ 51](#_Toc70622813)

[ANOTACE 52](#_Toc70622814)

[ANOTATION 53](#_Toc70622815)

Úvod

Ve své bakalářské práci se budu zabývat vývojem webové aplikace pro firmu Tesco SW. Tento program je určen pro vytváření struktury nové databáze nebo pro aktualizace stávajících verzí.

Téma jsem si zvolil, protože je úzce spojeno s mým studiem a vizí pracovní náplně v mé budoucí pracovní pozici. Dále také protože je u něj možné využít nových kreativních nápadů stejně tak jako i těch klasických.

Cílem mé bakalářské práce je vyvinout funkční aplikaci, která by sloužila specializovaným zaměstnancům firmy Tesco SW, a nahradit tak dosavadní nástroj, který je využíván pro již zmiňované generovaní struktur nových databází. Tento nástroj je již zastaralý a požadavky na jeho výstup se za léta užívání změnily. Byl tedy vznesen požadavek, aby se aplikace upravila do vyhovující podoby, což však nebylo možné vzhledem k tomu, že aktuální zdrojové kódy byly ztraceny. V tu chvíli bylo rozhodnuto vyvinout celou aplikaci znovu za pomoci reverzního inženýrství. Tímto způsobem byla provedena analýza chování aplikace, která vedla k zadání konkrétních požadavků pro vývoj nové aplikace.

Bakalářská práce se skládá z teoretické a praktické části. Teoretická část je dále rozdělena na sedm kapitol. V prvních dvou kapitolách je stručně představena původní aplikace, popis jejího fungování a zadaní podle kterého měla býtaplikace vyvíjena. V následujících kapitolách byly popsány použité technologie, programovací jazyk a další pojmy, které s tímto tématem souvisí. Například XML, což je značkovací jazyk, pomocí kterého jsou zapsána vstupní data do souborů, jenž se vkládají do aplikace. Tyto kapitoly byly sepsány za pomoci dostupných literárních a internetových zdrojů.

Praktická část je zaměřena již na samotný vývoj aplikace. V první kapitole je zmíněn proces reverzního inženýrství, díky kterému byly získávány další požadavky na aplikaci a poznatky, které byly klíčové pro její vývoj. V další kapitole je popsána struktura celého řešení. Jednotlivé podkapitoly pak popisují klíčové metody, které aplikace využívá k jejímu správnému fungování. Předposlední kapitola této práce již představuje finální verzi aplikace a jednotlivé kroky, které uživatel musí vykonat k dosažení výsledného skriptu. V poslední kapitole je pak popsán proces publikování a obfuskace aplikace.

1. Teoretická část
2. Představení původní aplikace

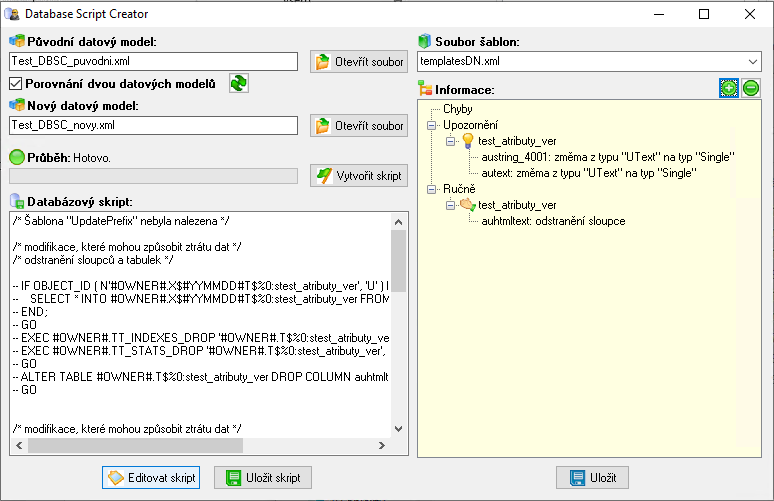
Původní aplikace Database Script Creator neboli DBSC byla napsána v programovacím jazyce Pascal, který je dnes považován za zastaralý. V rámci firmy se jedná o již historický nástroj z roku 2005, který je však stále denně využíván. V průběhu jejího působení ve firmě byla však několikrát modifikována, aby vyhovovala novým požadavkům od jejich uživatelů. Během této doby proběhlo mnoho migrací dat a různé úpravy v systému a došlo ke ztracení jejich aktuálních zdrojových kódů a zůstala pouze záloha z roku 2018, která již neobsahovala provedené modifikace a muselo se tedy této aplikaci přizpůsobovat například pozměněním šablon. To vedlo k požadavku tuto aplikaci znovu vytvořit, aby vyhovovala dnešním standardům firmy.

1.1 Popis fungování aplikace

Aplikace se využívá při vytváření struktury nové databáze (pro MS SQL Server a Oracle) nebo při aktualizacích databáze, jak již bylo zmíněno v úvodu této práce. Tento proces se vykonává na základě uživatelského vstupu, kterým je XML soubor, reprezentující cílový stav databáze nebo v případě porovnání dvou databází i jeho původní stav. Dále pak dojde ke zvolení příslušné šablony, podle které se skript vygeneruje. Výstupem je potom tedy SQL skript, který generuje databázové struktury (tabulky, view, trigerry, procedury, …), respektive mění stávající struktury v případě aktualizace.

V současné verzi se však již upustilo od používání různých šablon a používá se jen jedna výchozí, která obsahuje definice jak pro Oracle, tak i MSSQL a dále pak odkazuje na ostatní dodatečné skripty.

V průběhu generování tohoto skriptu se do vedlejšího panelu zapisují případné změny mezi původním a cílových stavem databáze. Po vykonání procesu generování, se skript zobrazí v náhledovém textovém bloku a na jeho konec se přidají dodatečné skripty, které jsou definované v šabloně s možností případné editace před samotným uložením. (viz Obrázek 1)



Obrázek 1 Původní DBSC, Zdroj: vlastní

**Procesu generování nové databáze nebo aktualizaci databáze tedy předchází tyto kroky:**

1. Designer (analytik) datového modelu vytvoří datový model a z něj potom XMI formu.
2. Designer databáze vytvoří šablony vhodné pro konkrétní druh a užití databáze na daném projektu (MSSQL/Oracle/SQLite a jiné specifické požadavky zákazníka).
3. Ze všech datových modelů, zúčastněných na projektu, vývojář vytvoří:
   1. definiční XML pro DBSC popisující celkovou strukturu databáze
   2. dodatečné SQL skripty, které obsahují (views, triggers, constraints, …)
4. Vývojář následně pomocí DBSC:
   1. vybere XML pro DBSC popisující cílový stav databáze
   2. vybere šablonu pro databázi
   3. volitelně vybere původní XML pro DBSC pro porovnání stavu databáze (tzv. rozdílový upgrade)
   4. spustí proces generování SQL upgrade scriptu
5. Vývojář v SQL upgrade skriptu ručně nahradí konstanty #OWNER# za schéma (MSSQL)/vlastníka schématu (ORACLE), ve které mají být objekty vytvářeny.
6. Vývojář spouští SQL upgrade skript na příslušném databázovém stroji.
7. Zadání

V návaznosti na předchozí kapitolu byly vytyčeny problémy, které by se měly výhledově řešit, ale nyní to není možné, protože neexistují zdrojové kódy k DBSC a nelze tedy proces dostatečně ovlivnit.

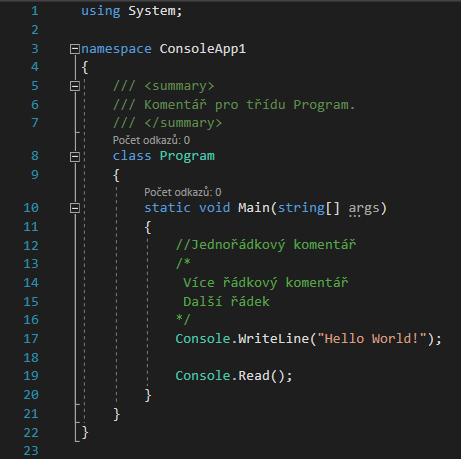
**Řešení problémů aktuálního procesu:**

1. Přidat mezikrok, kterým se zobrazí rozdíl při porovnání cílového stavu s původním nebo cílového s žádným.
   1. Odstraní se tak potřeba rozhodnutí, jestli se porovnávají dvě databáze nebo tvoří nová (aplikace toto sama vyhodnotí).
   2. Uživatel při kontrole také může sám rozhodnout, zda je opravdu potřeba generovat celý nový skript a může tak zamezit zbytečnému přidávání dodatečných skriptů. Jedná se totiž o velmi objemnou i časově náročnou část při upgradu databáze.
2. Úplně vynechat proces výběru šablony a nahradit ho strojovým nahráním výchozí šablony. Projektová šablona databáze je totiž soubor, který je přidružený k programu DBSC, logický má však být přidružen k projektu. K DBSC má být přidružena pouze výchozí šablona.
3. Vyvíjet aplikaci v jazyce C#, kvůli snadnějšímu pochopení programu pro ostatní pracovníky firmy.
4. Jazyk C#

Jedná se o objektově orientovaný programovací jazyk vyvíjený a představený společností Microsoft. Byl uveden jako součást vývojového prostředí .NET framework, které poskytuje služby pro své aktivní aplikace. Ačkoli vychází z programovacích jazyků C a C++, tak je v mnoha ohledech i blízko jednomu z dalších hojně užívaných jazyků a tím je Java.

Každý příkaz psaný v jazyce C# musí končit středníkem (;) nebo být vázán do bloků pomocí složených závorek ({}), může však být psán i přes několik řádků. Dále bylo pro psaní kódu sepsáno i několik konvencí, vzhledem k faktu, že tento jazyk rozeznává velká a malá písmena (je tzv. Case Sensitive). Z pravidla se jména tříd, rozhraní, balíků a mnoho dalších položek píše s velkým písmenem. Malým písmenem se na druhou stranu začínají třeba lokální proměnné nebo parametry. *[[1]](#footnote-1)*

Určitě je dobré mít nějakou představu o tom, jak daný programovací jazyk funguje. Na následujícím obrázku jsem pro názornou demonstraci vytvořil jednoduchou konzolovou aplikaci zobrazující ikonickou hlášku „Hello World!“ při jejím spuštění a nechá ji zobrazenou, dokud uživatel aplikaci neukončí nebo nestiskne enter.



Obrázek 2 Konzolová aplikace, Zdroj: vlastní

Prvních několik řádků souvisí se jmennými prostory, a to tím způsobem, že příkaz *using System;* implementuje knihovnu tříd, díky které můžeme zapisovat třídu jednoduchým zápisem *Console* namísto *System.Console*. Dalším příkazem *namespace ConsoleApp1* říkám, že daný kód je psán ve jmenném prostoru *ConsoleApp1* a pokud zde budeme chtít přistupovat k jinému jmennému prostoru (například *ConsoleApp2*), budeme tento jmenný prostor muset naimportovat stejným způsobem jako knihovnu tříd *System* a to tedy následovně *using ConsoleApp2*.

Dále lze v kódu vidět různé druhy komentářů. Prvním zmíněným je *summary*. Tento komentář se zobrazí při přesunutí kurzoru na název třídy, metody, Tasku atd., a měl by popisovat, co v tomto případě obsahuje třída *Program*. Jednořádkový komentář se využívá při popisu části kódu nebo k poznámkám. Obdobně se používá i víceřádkový komentář, kterým můžeme okomentovat třeba větší část kódu.

Každá spustitelná aplikace musí mít deklarovaný nějaký jeden vstupní bod a tím je metoda *Main()*, která je vyvolána jako první při spuštění aplikace. Příkaz *Console.WriteLine()* potom vypíše daný text do konzole a příkaz *Console.Read()* potéčeká na další uživatelský vstup, a tím tedy i nechává konzolové okénko aktivní. (viz Obrázek 2)

3.1 Základní prvky jazyka C#

**Třídy**

Jedním z klíčových prvků tohoto jazyka jsou třídy, které mohou obsahovat položky, metody, vlastnosti a další. U těchto jednotlivých položek pak lze použít i různé modifikátory, což jsou klíčová slova, která udávají informaci o jejich přístupu a mohou být *public, private, protected, internal* nebo případně *static.* Pokud takový modifikátor není uveden, je daný člen ze základu brán jako *private*. (Robinson, 2003)

**Konstruktory a destruktory**

Ke třídám se váží i konstruktory a destruktory. Každá má definovaný výchozí konstruktor, který je prázdný a bez parametrů, dá se ale dle potřeby upravit a každý nový objekt se pak vytváří pomocí operátoru *new.* Destruktor se potom liší od konstruktoru tím, že jeho název začíná vlnovkou (~) a teprve pak následuje jméno třídy. Dále nemůže mít parametry a nelze ho přetížit. Vzhledem k těmto jeho vlastnostem nemusí být definován. (Robinson, 2003)

**Dědičnost a polymorfismus**

Dědičnost v jazyce C# je upřesnění již existujících struktur a lze ji vyjádřit přímo v její definici pomocí dvojtečky, která se píše za jméno třídy (potomka) a následně je uveden název základní třídy (rodiče). Potomek potom dědí vlastnosti i schopnosti svého rodiče a může je i upravovat.

Polymorfismus poté umožňuje používat jednotné rozhraní pro práci s různými typy objektů. Umožňuje například objektům volat metody se stejným názvem, ale jinou implementací. Například třída *Cat* a *Dog* dědí z třídy *Pet,* která má metodu *MakeSound()*, metoda pro třídu *Dog* vypíše „*Haf Haf“* a pro třídu *Cat* zase *„Mňau“.* Typicky tedy umožňuje volat metodu s různými parametry, přetěžovat operátory nebo přepisovat poděděné metody.

1. Extensible Markup Language

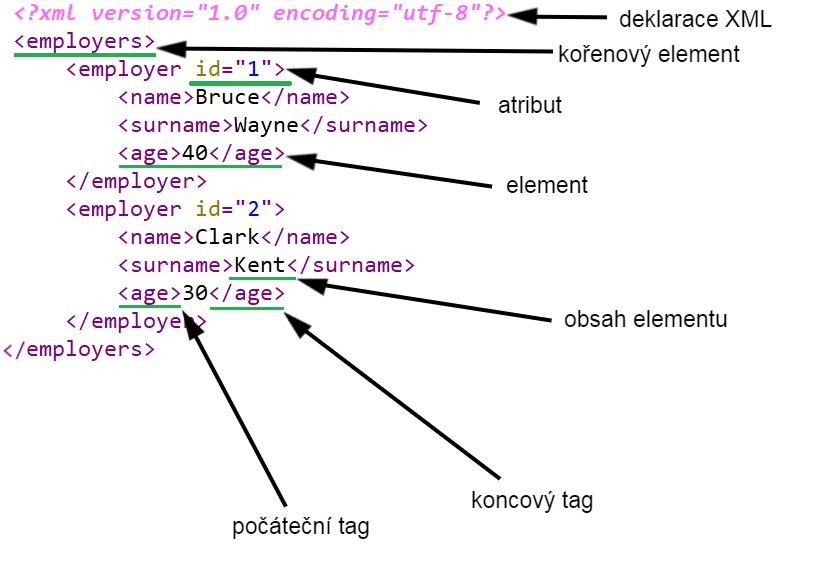
XML je zkratkou pro eXtensible Markup Language a jedná se o značkovací jazyk podobný například HTML. Slouží k snadnému uchovávání a přenášení dat, avšak samotné XML nic nedělá, a je tedy nutné k němu napsat kus kódu, který požadovaná data zobrazí v čitelnější podobě. (Kosek, 2009)

XML nepoužívá předdefinované tagy. Autor si je sám vytváří v průběhu psaní v návaznosti na jeho konkrétní úmysl narozdíl například od již zmíněného HTML, kde jsou tagy již definované a mají svůj konkrétní význam.

4.1 Struktura XML dokumentu

Každý XML dokument by měl začínat svou deklarací, ve které se určuje, jakou verzi XML používá a případně i jeho encoding. Od ostatních tagů se liší tím, že kromě ostrých závorek je ohraničena i otazníky.

Základem jsou jeho elementy, které jsou ohraničeny tagy a mohou obsahovat atributy, které představují doplňující informace. Veškeré tagy jsou case-sensitive (citlivé na velikost písmen), nemohou obsahovat mezery a první znak musí být písmeno nebo podtržítko. Jinak může být jejich název naprosto libovolný a mohou obsahovat například azbuku nebo české znaky.



Obrázek 3 XML vysvětlivky, Zdroj: vlastní

Takový formát však obsahuje i bílé znaky, které jsou součástí dokumentu a je s nimi tedy nutno počítat i při zpracování. Tyto bílé znaky se mohou objevovat v různých formách (mezery, konce řádků nebo tabulátory) a slouží k zpřehlednění zápisu. Zpravidla se každý nový element uvádí na nový řádek s odsazením odpovídajícím jeho zanoření. (viz Obrázek 3)

Také lze vkládat komentáře, pokud je potřeba něco vysvětlit nebo dočasně skrýt. Ačkoli jsou komentáře součástí dokumentu, netvoří součást programového kódu a to znamená, že jej programy nezpracovávají. Mohou obsahovat cokoli, kromě posloupnosti spojovníků. Zapisují se mezi znaky ve tvaru <!-- komentář -->. (Kosek, 2009)

Pokud je potřeba zapsat speciální znaky, které by působily problémy například v zápisu značkování, používají se tzv. znakové entity. V XML je předdefinováno celkem 5 znakových entit, a to konkrétně pro:

* < ve tvaru &lt;
* > ve tvaru &gt;
* & ve tvaru &amp;
* ´ ve tvaru &apos;
* „ ve tvaru #quot;

To však nijak neomezuje použití ostatních znaků, které se dají vložit pomocí Unicode kódu. Před vlastní kód je však nutné uvést prefix, který je pro decimální kód ve tvaru &#dddd; a pro hexadecimální kód ve tvaru &#xhhhh;. Takže symbol „menší než“ byl zapsán v decimálním kódu jako &#60; a v hexadecimálním jako &#x3C;. (Kosek, 2009)

4.2 Document Type Definition

Definice typu dokumentu neboli DTD je užitečnou osnovou pro XML, která umožňuje hlídat správnou strukturu dokumentu tím, že definuje vzájemné vztahy mezi jednotlivými elementy nebo jaké elementy a atributy se mohou používat.

V závislosti na potřebách uživatele nebo firmy se používají různá DTD například pro technické dokumentace. Pokud je určité DTD nevhodné pro dané požadavky, může se↓napsat úplně nové, které vyhovuje určitým standardům.

Přidává se hned po deklaraci XML buď jako odkaz na samostatný soubor, nebo případně rovnou jako součást dokumentu (podobně jak kaskádové styly pro HTML). Odkaz na soubor by tedy byl zapsán následovně <!DOCTYPE kořenový\_element SYSTEM „URL\_souboru“>. (Kosek, 2009)

4.3 Zpracování XML dokumentu

Ke zpracování XML dokumentu se používá například XML serializér, který disponuje funkcí jak serializace, tak deserializace. Metoda serializace tedy může vzít zadaný objekt, kterým může být například třída, a zapíše jej v XML formátu do souboru. Naopak deserializací docílíme opačného výsledku, kdy se této metodě předá parametr ve formě streamu nebo třeba souboru a ta převede data do podoby, se kterou je možno dál pracovat.

Pokud je však deserializační metoda z nějakého důvodu nevyhovující, lze ji nahradit vlastním řešením. Takovým řešením může být třeba asynchronní operace, která pomocí XmlReaderu iteruje skrz celý dokument nebo textový řetězec a jednotlivé hodnoty přidává do třídy. Tento nástroj umožňuje i definovat vlastní nastavení, které může třeba ignorovat bílé znaky. (viz Zdrojový kód 1)

XmlReader funguje jako čtecí nástroj pouze pro data XML nebo v datovém proudu, takže v případě uložení dat v podobě textového řetězce bylo nutné tyto data předat jako instanci StringReaderu, kterou už byl schopen přijmout. Umožňuje procházet daty a číst obsahy jednotlivých uzlů. Jeho vlastnosti vždy odpovídají vlastnostem daného uzlu, ve kterém se nachází. Umožňuje též provádět kontrolu shody dat a ověřovat je oproti DTD nebo schématu. Metodou *Create* se tedy vytvořil nový Reader, jehož vlastnosti se mohly postupně naplňovat do požadovaného modelu pro další práci s těmito daty.[[2]](#footnote-2)

|  |
| --- |
| public async Task<DbModel> CreateDbModelAsync(string xmlContent)  {  return await Task.Run(async () =>  {  var dbModel = new DbModel();  var settings = new XmlReaderSettings  {  Async = true,  IgnoreWhitespace = true,  };  var reader = XmlReader.Create(new StringReader(xmlContent), settings);  await AppendModel(dbModel, reader);  return dbModel;  });  } |

Zdrojový kód 1 Deserializace XML, Zdroj: vlastní

1. Reverzní inženýrství

Reverzní inženýrství nebo také reengineering se dá chápat jako proces zpětné analýzy nějakého programu s cílem zjistit jeho komponenty, vazby mezi nimi a následně vytvořit software obdobný nebo na vyšší úrovni. Při tomto procesu se zkoumají činnosti, které program vykonává a postupně jsou z nich tvořeny požadavky na jejich zrekonstruování.

1. Volba frameworku pro uživatelské rozhraní

Sice se jedná o rozhodnutí spíše v pozdějších fázích vývoje, ale hned na začátek bych rád uvedl problematiku výběru vyhovujícího frameworku pro vytvoření uživatelského rozhraní neboli UI. V úvahu připadly dvě varianty, které bylo nutné otestovat a na základě jejich vlastností se poté rozhodnout.

6.1 Blazor

Jedná se o relativně nový open-sourcový webový framework vyvíjený společností Microsoft. Vývojářům umožňuje zejména tvořit tzv. single-page aplikace, „*které pracují přímo uvnitř prohlížeče a nepotřebují tedy znovu načítat stránky během jejich používání.“*[[3]](#footnote-3) Blazor také dokáže komunikovat s JavaScriptem, který se běžně využívá při tvorbě webových stránek a je často vkládán přímo jako součást HTML. Vychází z Razor Pages, čemuž napovídá i jeho jméno, které vzniklo „mutací“ slov Browser a Razor. [[4]](#footnote-4)

6.1.1 Rozdíl mezi Razor Pages a Blazorem

Mnoho vývojářů mluví o Razor Pages a Blazoru v úzkém spojení a někteří tyto dvě technologie ani nerozlišují. Důvodem je totiž fakt, že Blazor používá Razorové komponenty k vytvoření aplikace. Oba frameworky však mají stejnou syntaxi a používají symbol zavináče @ pro přechod z HTML formátu na C#. Tímto způsobem je pak schopen zobrazovat například hodnoty proměnných nebo používat podmínky a cykly.

Blazor se však liší v jedné zásadní věci, a to v tom, že je frameworkem pro single-page aplikace, jak již bylo zmíněno v úvodu této kapitoly. Funguje tedy tak, že první požadavek na server vrátí celou aplikaci, včetně všech definovaných komponent. Ty mohou reprezentovat třeba celou stránku nebo jen její malé části. Jsou responzivní a mohou tedy být skryty, zobrazeny nebo aktualizovány v závislosti na datech nebo událostech.

Razor Pages se vždy zaměřovaly na podávání celých stránek do prohlížeče. Což znamená, že kdykoli uživatel provedl nějakou akci, poslal se požadavek na server a ten poslal zpět data, která celou stránku vyrenderovala znovu.[[5]](#footnote-5)

**Hlavní výhody:**

* Práce v jazyce C# namísto JavaScriptu
* Poměrně malá velikost oproti Electronu
* Možnost spuštění v rámci serveru nebo přímo v prohlížeči klienta díky Web Assembly, což je instrukce v binárním formátu poskytující kompilaci programovacích jazyků jako je například C#.

**Komponenta Blazoru se skládá ze dvou částí:**

1. Sekce kde uživatel definuje vzhled stránky pomocí HTML kódu a kaskádových stylů. V této části se též užívá výše zmíněný symbol zavináče a definují se zde i používané namespacy či používané služby pomocí příkazů @inject a @using. (viz Zdrojový kód 2)
2. Druhá část je vymezena pro C# kód, kterou je možno využít například pro definování proměnných nebo pro práci s dokumenty v případě jejich nahrání. Je ohraničena složenými závorkami za příkazem @code.[[6]](#footnote-6)

(viz Zdrojový kód 3)

|  |
| --- |
| <div class="grid-container2">  <textarea class="center output item4\_1" **@bind**="finalScript" **@bind:event**="oninput">  @finalScript  </textarea>  <textarea class="center output item4" **@bind**="resultDiff" **@bind:event**="oninput" readonly>  @resultDiff  </textarea>  <button class="btn btn-primary item5" **@onclick**="GenerateScript"  disabled="@(!(models?.ModelDiff != null && templateStream.Length!=0))">  Generate script  </button>  <button class="btn btn-primary item6" **@onclick**="SaveScript"  disabled="@(string.IsNullOrEmpty(finalScript))">  Save script  </button>  </div> |

Zdrojový kód 2 Ukázka kódu Blazor, Zdroj: vlastní

|  |
| --- |
| @code  {  IFileListEntry filePrevious;  IFileListEntry fileTempPrev = null;  string previousContent;  IFileListEntry fileNext;  IFileListEntry fileTempNext = null;  string nextContent;  IFileListEntry template;  MemoryStream templateStream = new MemoryStream();  IDbModelComparer dbModelComparer;  DbscModels models = new DbscModels(null, null, null);  StringBuilder diff = new StringBuilder();  string resultDiff = string.Empty;  string finalScript = string.Empty;  protected override async Task OnInitializedAsync()  {  var response = await Http.GetAsync("api/HomeController/" + "templatesDN.xml");  var stream = await response.Content.ReadAsStreamAsync();  stream.Position = 0;  await stream.CopyToAsync(templateStream);  templateStream.Position = 0;  }  } |

Zdrojový kód 3 Ukázka části code v Blazoru, Zdroj: vlastní

Názornou ukázku praktického použití Blazoru lze vidět na výše uvedeném obrázku. Jak již bylo uvedeno, tak v první části je psáno HTML, tedy vzhled stránky. V tomto konkrétním příkladu jsem vytvářel div pro zobrazení dvou textových oblastí (textarea) společně s tlačítky pro provedení příslušné operace. V části *@code* jsou deklarované proměnné, které jsem v této části používal a ukázka asynchronní operace, vykonané při načtení stránky.

Obsahem zmíněných textových polí bude *finalScript* a *resultDiff*. To jsou proměnné, které byly naplněny textovým řetězcem v průběhu chodu aplikace po vykonání určité akce uživatelem. Pomocí příkazu *bind* jsem zajistil pravidelnou aktualizaci těchto textových polí, pokud se v proměnných stala nějaká změna. *Bind:event* na druhou stranu aktualizuje dané proměnné na základě uživatelského vstupu, tzn. pokud uživatel udělal změnu v jednom z polí, provede se tato změna i na proměnné. Příkazem *@onclick* říkám, že pokud uživatel klikne na tlačítko, zavolá se požadovaná metoda z kódové části. Tyto tlačítka mají i atribut *disabled*, který obsahuje podmínku, která říká, že když je pravdivá, tedy má hodnotu *true*, tak se tlačítko pro používání znepřístupní. Konkrétně v tomto případě to znamená, že pokud proměnná *finalScript* nemá žádnou hodnotu, bude toto tlačítko pro uložení znepřístupněno.

6.2 Electron

Podobně jako Blazor je Electron open-sourcovým frameworkem, avšak narozdíl od něj využívá JavaScriptu místo C#. Takže za pomoci HTML, CSS a JavaScriptu umožňuje tvorbu nativních multiplatformních desktopových aplikací.

**Hlavní výhody:**

* Podpora napříč prohlížeči
* Aplikace se nemusí psát pro různé platformy, je napsána jednou a pak je zabalena do instalačního balíčku pro konkrétní platformu.[[7]](#footnote-7)

**Hlavní nevýhody:**

* Výsledný instalační balíček může mít několik MB (základní demo aplikace měla okolo 130 MB).
* Pokud aplikace pracuje s velkým množstvím dat (což je zrovna případ DBSC), tak může mít problémy s výkonem.[[8]](#footnote-8)

6.3 Vyhodnocení

Na základě výše zmíněných pro i proti bylo usneseno, že Electron se nehodí jako ideální framework pro aplikaci DBSC. Hlavním faktorem pro toto rozhodnutí byl fakt, že DBSC pracuje s obrovským množstvím dat, kdy výsledný soubor dosahuje až 65 MB. Délka zpracování souborů a generování výsledného skriptu by potom tedy bylo víc časově náročné, než bylo původně odhadováno.

Velikost výsledného instalačního balíčku byla zbytečně velká už jen při samotné demo verzi, vzhledem k obsáhlosti projektu by potom výsledný balíček mohl být příliš velký, a tím by byla ztížena jeho implementace k uživatelům.

1. Databáze

Pod pojmem databáze je možné si představit soubor, který obsahuje uspořádaná data, s kterými se dá efektivně manipulovat a je v elektronické podobě uložen v počítačovém systému. K jeho řízení se používá DBMS neboli systém pro správu databáze. Data obsažená v databázi jsou obvykle modelována jako tabulky složené ze sloupců a řádků. Takový model umožňuje efektivní zpracování a vytváření dotazů, díky kterým je umožněn snadný přístup k datům, úpravám, řízení a organizaci dat. (Oppel, 2006)

7.1 Typy databází

Existuje několik variant databází. Jejich použití záleží na rozhodnutí firmy, jakým způsobem chce s daty pracovat. Rád bych zde pro představu uvedl pár nejzákladnějších typů databází:

* **Relační model –** V dnešní době nejpoužívanější typ databáze. Data má organizovaná v tabulkách a v nich jsou prováděny i všechny databázové operace. Začala být dominantní v 80. letech 20. století jako nástupce síťového typu.
* **Hierarchický model –** Dnes již značně zastaralý typ pocházející ze 70. let 20. století. Data jsou organizována do stromové struktury a každý záznam zde představuje určitý uzel ve vztahu rodič-potomek, kde jeden rodič může mít více potomků, ale každé dítě jen jednoho rodiče.
* **Síťový model –** Do povědomí se dostal jako nástupce hierarchického typu, ale hodně rychle ho nahradil typ relační databáze. Jedná se vlastně o zobecnění hierarchického typu, který jej obohatil o mnohonásobné vztahy.**[[9]](#footnote-9)**

7.2 Structured Query Language

Structured Query Language neboli SQL je standardizovaný dotazovací jazyk, který používají skoro všechny databázové systémy. V různých databázových systémech se může mírně lišit jeho implementace a případně obsahovat i určitá rozšíření, jako například vlastní datové typy nebo funkce. Řadí se také mezi deklarativní jazyky, což ve zkratce znamená, že zadavatel nespecifikuje konkrétní postup, ale pouze popíše požadovaný výsledek.

|  |
| --- |
| --zobrazte všechny filmy s názvem "The Avengers„ a seřaďte je podle data vydání vzestupně  SELECT \* FROM Film WHERE Nazev = 'The Avengers' ORDER BY DatumVydani  --zobrazte unikátní hodnoty atributu Jmeno v tabulce osob a seřaďte je vzestupně  SELECT Jmeno FROM Osoba WHERE Jmeno IS NOT NULL ORDER BY Jmeno  --zobrazte deset nejdelších filmů, seřaďte dle délky trvání sestupně  SELECT TOP 10 \* FROM Film ORDER BY Trvani DESC  --zobrazte filmy, které nemají uvedeno datum vydání  SELECT \* FROM Film WHERE DatumVydani IS NULL |

Zdrojový kód 4 Ukázka kódu SQL, Zdroj: vlastní

Na výše uvedeném kódu je uvedeno pár základních dotazů pro zobrazení požadovaných dat z databáze pomocí příkazu *SELECT*. Co bude výstupem je již jasné z komentářů uvedených v obrázku, ale pro názornost si dovolím podrobněji popsat logiku jejich zápisu.

V prvním příkladu si pomocí \* vyberu všechny vlastnosti z tabulky *Film*, u kterých se jejich atribut *Nazev* rovná požadovanému textovému řetězci a jako poslední se tato data seřadí podle atributu *DatumVydani*. Jak je z příkladu patrné, tak základní syntaxe není vůbec složitá, jednoduše tvoříme „věty“ pro dosažení požadovaného výsledku.

Jazyk SQL dále umožňuje vytvářet tabulky či přidávat, upravovat nebo mazat data v databázi. Díky těmto vlastnostem je též rozdělen do kategorií podle jejich funkčnosti:

* **DQL (jazyk pro dotazování) –** Zde patří příkaz SELECT a jeho klauzule
* **DML (jazyk pro manipulaci) –** Obsahuje příkazy pro manipulaci s daty jako například INSERT, DELETE nebo UPDATE
* **DDL (jazyk pro definování) –** Příkazy v této kategorii zajišťují úpravu nebo vytváření nových struktur pomocí příkazů CREATE, ALTER, DROP
* **DCL (jazyk pro řízení) –** Slouží k provozu a údržbě databáze s příkazy jakou jsou např. GRANT, COMMIT a další [[10]](#footnote-10)

7.3 Systém řízení báze dat (SŘBD)

SŘBD je programový systém, který zajišťuje veškeré operace, které se s daty provádějí, a slouží též i jako takový prostředník pro přístup k datům pomocí aplikací. Existuje mnoho takových systémů, jako je např. MSSQL, Oracle, MySQL, MS Access atd., ve firmě TescoSW se však nejčastěji využívá řešení Oracle a MSSQL.[[11]](#footnote-11)

Asi nejzásadnějším rozdílem je fakt, že MSSQL lze nainstalovat pouze na Windows serverech, kdežto řešení od společnosti Oracle takové omezení nemá a je spustitelné na libovolném OS. Dále stojí za zmínku, že Oracle je open-sourcovým systémem, narozdíl od MSSQL, které nabízí pouze zkušební verzi zdarma a jeho další používaní je zpoplatněné. Syntaxe užívaná v Oracle je sice komplexnější než v MSSQL, ale je také mnohem efektivnější.[[12]](#footnote-12)

7.4 Datový model

Datový model se dá chápat jako plán, který popisuje všechny objekty, procesy, atributy a vztahy mezi nimi. Takový datový model je potom již zmiňovaným vstupním souborem do aplikace DBSC. (Hernandez, 2006)

**Entity a atributy**

Entita je základním prvkem datového modelu. Dá se chápat jako samostatný objekt, který je existenčně nezávislý na ostatních entitách. K entitám se váže pojem atribut, ten reprezentuje konkrétní vlastnosti entity a tím ji podrobněji popisuje. (Hernandez, 2006)

**Relace**

Relace je chápána jako vazba mezi entitami. Je to tedy popis, jak mohou být dvě nebo více entit spojeny. Obvykle bývají vyjádřeny jako slovesa a rozlišuje se i několik typů:

* 1:1 – jednomu záznamu z první tabulky odpovídá maximálně jeden záznam z druhé
* 1:N – jednomu záznamu z první tabulky může odpovídat libovolný počet záznamů z druhé
* M:N – libovolnému počtu záznamů z první tabulky může odpovídat libovolný počet záznamů z druhé[[13]](#footnote-13)

1. Praktická část
2. Reengineering původní aplikace

Celá analýza spočívala ve výběru činnosti, prováděné aplikací, a následné porozumění všem procesům, které vedly k jejímu úspěšnému vykonání. Prvně jsem se zaměřil na uživatelské vstupy a dále na proces generování databáze a výstup aplikace.

8.1 Uživatelský vstup

V této části jsem se zabýval pochopením struktury vstupních dokumentů zadávané uživatelem do aplikace. Jedná se o dokumenty XML popisující strukturu cílového nebo původního stavu databáze. Struktura je vždy stejná, a když pominu deklaraci XML, tak je tvořena kořenový elementem *Model*, který obsahoval další jednotlivé elementy popisující konkrétní tabulky, přidružené soubory či typ používané šablony.

**Hlavními elementy v dokumentu jsou:**

* **SourceDriver** *–* který udává typ používané šablony
* **AdditionalScripts** *–* což jsou uživatelské skripty přidané na konec generování výsledného skriptu
* **Object** *–* reprezentuje element obsahující informace o dané tabulce databáze
* **Attribute***–* obsahuje informace o řádku tabulky

Tyto elementy potom obsahují atributy, které definují jejich vlastnosti (název, datový typ…). Atribut *xmlns:xsi* potom říká, že elementy a datové typy obsažené ve schématu pocházejí ze jmenného prostoru uvedeném v uvozovkách. (viz Zdrojový kód 5)

|  |
| --- |
| <?xml version="1.0"?>  <Model Name="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <SourceDriver xsi:type="adoMSSQL">  </SourceDriver>  <AdditionalScripts>  <additionalscript name="TestRemoved" path="D:\Removed.sql" forupdate="true" />  <additionalscript name="TestChanged1" path="D:\Changed.sql" forupdate="false" />  </AdditionalScripts>  <Object Name="Test" ClassName="Test" SourceName="test" Version="true">  <Attribute xsi:type="BCD" Name="ID" SourceName="ID" IsObjectID="true"></Attribute>  <Attribute xsi:type="String" Size="64" Name="TST" SourceName="tst" IsObjectID="false"></Attribute>  </Object>  </Model> |

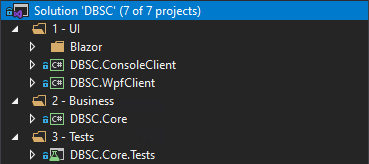
Zdrojový kód 5 Ořezaná ukázka vstupního XML, Zdroj: vlastní

8.2 Proces generování databáze

Díky dochovaným zálohám staré aplikace jsem měl možnost nahlédnout na samotný kód a lépe tak pochopit celou její funkčnost. Pomocí debuggeru, což je účinný nástroj pro krokování jednotlivých částí aplikace, jsem byl schopen popsat s jakými konkrétními daty aplikace pracuje, aby dosáhla požadovaného výsledku.

1. Struktura řešení

Vytvořené řešení (solution) obsahuje celkem 7 projektů, které vždy začínají slovem *DBSC* a jsou rozděleny do 3 hlavních části pro lepší orientaci ostatním programátorům, a i pro mě samotného. Tyto části jsou tedy uživatelské rozhraní (UI), hlavní logika aplikace (Business) a testování (Tests). (viz Obrázek 4)



Obrázek 4 Struktura projektu, Zdroj: vlastní

9.1 User Interface

Tato sekce obsahuje projekty týkající se uživatelského rozhraní. Jak je patrné z výše uvedeného obrázku, tak Blazor není jediným uvedeným řešením pro UI. Ostatní zmíněné projekty jsou však plánovány až do budoucna a jsou zde zatím uchovány, aby nebyly zapomenuty. Například *ConsoleClient* je zamýšlen jako konzolová aplikace pro zkušenější uživatele, kteří by díky ní nemuseli spouštět celou aplikaci a kontrolovat případné rozdíly v datových modelech. Namísto toho by jim bylo umožněno rovnou zadat cestu k požadovaným souborům a výsledkem by byl již vygenerovaný skript.

**Blazor se v tomto řešení skládá ze tří projektů:**

* **Client** – tento projekt obsahuje komponenty, které vytváří vizuální prostředí.
* **Server** – je určen řadičům a statickým souborům.
* **Shared** – obsahuje třídu pro uložení informací o šablonových souborech.

9.1.1 Client

Klientská část Blazoru obsahuje stránky spustitelné v rámci webové aplikace a funkce, díky kterým uživatel volá metody z logické části řešení.

Hlavní stránkou je *Index.razor*, která se zobrazí při spuštění. V teoretické části jsem tuto stránku popsal spíše z hlediska HTML, proto bych se zde rád zaměřil na část *@code*, ve které se využívá jazyk C#.

Hned na začátek bych rád zmínil asynchronní operaci, která při načtení komponenty nahraje šablonový soubor do streamu, který se pak předává jako parametr do metody zajišťující proces generování.

|  |
| --- |
| protected override async Task OnInitializedAsync()  {  var response = await Http.GetAsync("api/HomeController/" + "templatesDN.xml");  var stream = await response.Content.ReadAsStreamAsync();  stream.Position = 0;  await stream.CopyToAsync(templateStream);  templateStream.Position = 0;  } |

Zdrojový kód 6 Funkce při spuštění aplikace, Zdroj: vlastní

Tato operace je již v Blazoru definována a je tedy potřeba ji přepsat pomocí klíčového slova *override*. Jako první se pomocí třídy HttpClient (Http) zavolá metoda *GetAsync*, která pošle požadavek GET na řadič v serverové části Blazoru (viz Kapitola 9.1.2) a vrátí obsah šablonového souboru jako *HttpResponseMessage*. Obsah odpovědi na tento požadavek se poté přečte jako datový stream pomocí metody *ReadAsStreamAsync* a následně je třeba nastavit pozici tohoto streamu na počáteční hodnotu (jelikož metoda pro čtení prochází celým streamem a po dokončení nechává jeho pozici na posledním přečteném bodě). Poté je potřeba jej nahrát do již předpřipraveného *templateStream*, který je datového typu *MemoryStream* a opět nastavit jeho pozici na začátek. Při čtení streamu se totiž začíná od jeho pozice, a pokud je nastavená na koncoví bod, tak potom metoda pro jeho čtení nemá co číst. (viz Zdrojový kód 6)

|  |
| --- |
| public async void SaveScript()  {  var fileName = ScriptComposer.GetFileName(filePrevious?.Name, fileNext?.Name);  if (string.IsNullOrEmpty(fileName))  fileName = "default.txt";  var bytes = System.Text.Encoding.UTF8.GetBytes(finalScript);  await FileUtil.SaveAs(js, fileName, bytes);  } |

Zdrojový kód 7 Metoda pro uložení skriptu, Zdroj: vlastní

V metodě *SaveScript* (viz Zdrojový kód 7) definuji jméno ukládaného souboru, které má stejný název jako vstupní soubor pro cílovou databázi. Pokud však nejsou nahrány žádné soubory, tak se soubor uloží s názvem *default.txt.* Vygenerovaný skript v textovém řetězci se nyní musí převést na sekvenci bajtů. Ty jsou v dalším kroku předány dál a o samotné uložení se postará JavaScriptová funkce *saveAsFile*, která jevyvolaná pomocí asynchronní operace *InvokeAsync*. (viz Zdrojový kód 8)

|  |
| --- |
| public static class FileUtil  {  public async static Task SaveAs(IJSRuntime js, string fileName, byte[] data)  {  await js.InvokeAsync<object>(  "saveAsFile",  fileName,  Convert.ToBase64String(data));  }  } |

Zdrojový kód 8 Volání JavaScriptové metody, Zdroj: vlastní

Na dalším obrázku bych rád zmínil poslední metodu, která se stará o nahrání vstupních souborů a jejich následné porovnání. Během tohoto procesu se také postupně sbírají data do *MessageBufferu*, který je součástí třídy rozdílového modelu v třídě *DbscModels,* z něj se potom vypisují rozdílové hlášky. První je v metodě ošetřena přítomnost vstupních souborů pomocí podmínky *if*. Pokud políčko pro nahrání souboru není prázdné a zároveň se nerovná souboru, který byl případně už předtím nahrán (*fileTempPrev* a *fileTempNext*), tak nahraje jeho obsah do textového řetězce (*previousContent* a *nextContent).* Poté se obdobnou podmínkou zamezí zbytečnému generování rozdílového souboru a hlášek v případě, že jsou nahrané soubory stále stejné.

*ResultDiff* slouží jako proměnná pro zobrazení výsledných rozdílových hlášek a *diff* potom slouží jako *StringBuilder*, který zlepšuje výkon při procházení několika textových řetězců v cyklech *foreach* (viz Zdrojový kód 9). *Models* je již instancí třídy *DbscModels* s naplněnými hodnotami, a je tedy možné je dál použít pro vypsání hlášek.

V cyklu *forech* postupně procházím hodnotami *MessageBufferu*, který má v sobě uloženy hlášky, rozdělené podle elementu, pro který se vypisují. Postupně se tedy hlášky přidávají do *StringBuilderu* a na samém závěru jej převedu na textový řetězec, abych ho mohl zobrazit v textové oblasti.

|  |
| --- |
| private async Task GetModelDiff()  {  if (filePrevious != null && fileTempPrev != filePrevious)  previousContent = await GetFileContent(filePrevious);  if (fileNext != null && fileTempNext != fileNext)  nextContent = await GetFileContent(fileNext);  if (fileTempPrev != filePrevious || fileTempNext != fileNext)  {  resultDiff = string.Empty;  diff = new StringBuilder();  models = await Dbsc.GetDbDiff(nextContent, previousContent, new DbModelParser());  foreach (var messages in models.ModelDiff.MessageBuffer.Values)  {  diff.Append(messages.RootMessage);  foreach (var changedObj in messages.ObjectChanges)  {  diff.Append(changedObj);  }  foreach (var changedAdditional in messages.AdditionalScriptChanges)...  foreach (var attr in messages.AttributeMessages)...  foreach (var changedAttr in messages.ChangedAttributeMessages.Values)...  {  foreach (var item in changedAttr)...  }  foreach (var encrypt in messages.EncryptedMessages)...  foreach (var extra in messages.ExtraMessages)...  foreach (var inter in messages.InterfaceMessages)...  }  }  resultDiff = diff.ToString();  fileTempPrev = filePrevious;  fileTempNext = fileNext;  } |

Zdrojový kód 9 Metoda pro vytvoření rozdílového souboru a hlášek, Zdroj: vlastní

9.1.1 Server

V serverové části klienta Blazoru bych rád zmínil sekci s řadičem (*Controllers),* který zajišťuje komunikaci se serverovou částí.

Třídu řadiče ještě blíže specifikuji za použití dědičnosti z třídy *Controller* a atributů *ApiController* a *Route,* který mu udává cestu pro jeho volání*.* Funkce v něm obsažená se volá pomocí metody *HTTP* *GET*, kdy se uvede název řadiče a název souboru, který si chci nahrát (viz Zdrojový kód 6). V samotné funkci se potom zadá cesta k umístění požadovaného souboru pomocí *IWebHostingEnvironment*, díky kterému můžu přistupovat ke složkám v serverové části a za použití parametru *Name* předaného z klientské části. (viz Zdrojový kód 10).

|  |
| --- |
| [ApiController]  [Route("api/HomeController")]  public class HomeController : Controller  {  private IWebHostEnvironment \_env;  public HomeController(IWebHostEnvironment env)  {  \_env = env;  }  [HttpGet("{Name}")]  public IActionResult Download(string Name)  {  var bytes = System.IO.File.ReadAllBytes(\_env.ContentRootPath + @"\StaticFiles\" + Name);  return File(bytes, "text/xml");  }  } |

Zdrojový kód 10 Řadič, Zdroj: vlastní

9.2 Business

Business část představuje logiku celé aplikace. Obsahuje vlastní řešení zásadních prvků, které jsou rozděleny do složek podle jejich funkcí.

* **Comparer** – obsahuje funkci pro porovnání vstupního souboru (vstupních souborů) a třídy reprezentující rozdílový datový model, kam se tyto rozdíly ukládají.
* **DatabaseModel** – zde jsou třídy, ze kterých se skládá vstupní databázový model (každá třída reprezentuje skupinu elementů ze vstupního XML).
* **DiffMessages –** tato část uchovává třídy potřebné pro vytvoření rozdílových hlášek.
* **Parsers –** obsahují metody pro deserializací XML souborů (datových modelů i šablony) a zároveň slouží i jako parsery, které ověřují jejich správnou strukturu.
* **TemplateModel –** podobně jak *DatabaseModel* tato část obsahuje třídy, které reprezentují jednotlivé elementy z šablonového souboru.

V části *Comparer* a *Parsers* je interface neboli rozhraní, které slouží k definování funkcionality metod bez jejich implementace. Obsahuje variantu pro práci se vstupními daty ve formě cesty k souboru (původně vytvořeno kvůli testování, nyní je předpřipraveno pro konzolovou aplikaci), obsahu uloženém v textovém řetězci nebo v podobě streamu.

Dále jsou v Business části projektu obsaženy tyto třídy:

* **Dbsc –** je hlavní třídou, která obsahuje Task zajišťující generovaní výsledného skriptu.
* **DbscModels –** reprezentuje společnou třídu pro všechny modely (cílový model, rozdílový model a případně původní model).
* **ScriptComposer –** tato třída již kompletuje samotný skript na základě vstupních dat.

9.2.1 Comparer

V této části jsou uloženy třídy reprezentující rozdílový model, který obsahuje veškeré rozdíly ve vstupních souborech. Každá změna je pak zaznamenána s její původní a novou hodnotou, tak aby se s těmito změnami dalo dále pracovat.

Metoda pro porovnání dvou souborů v tomto projektu funguje tak, že vytvářím novou instanci rozdílového modelu, kterou potom vracím. Rozdílový model přijímá 3 parametry, a to třídu pro uchování rozdílových hlášek, datový model, reprezentující cílový stav databáze, a případně i jeho původní stav. (viz Zdrojový kód 11)

|  |
| --- |
| public DbChangedModel CompareModel(DbModel previousModel, DbModel nextModel)  {  var treeMessages = new MessagesTree();  var dbDiff = new DbChangedModel(previousModel, nextModel, treeMessages);  return dbDiff;  } |

Zdrojový kód 11 Metoda CompareModel, Zdroj: vlastní

9.2.2 DiffMessages

Pro záznam rozdílových hlášek byla vytvořena třída *Messages* a dále třída *MessagesTree,* která obsahuje přetížené varianty metody *Add* pro přidání konkrétních hlášek k jednotlivým elementům. (viz Zdrojový kód 12)

Tato metoda, stejně jako její přetížené varianty, funguje tak, že je definován hlavní klíč (*keyObject*), který je stálým bodem pro přidávání dalších záznamů. Pokud se například udála změna ve dvou elementech *Attribute*, které jsou subelementy hlavního elementu *Object*, tak budou mít oba stejný klíčový objekt a obě rozdílové hlášky se tedy zapíší do stejného slovníku.

|  |
| --- |
| /// <summary>  /// Adds given message whether was element Object added or removed.  /// </summary>  /// <param name="keyObject">Value of Object.</param>  /// <param name="rootMessage">Given message.</param>  public void Add(DbObject keyObject, string rootMessage)  {  var key = keyObject.SourceName;  if (!ContainsKey(key))  Add(key, new Messages());  (this[key]).RootMessage = rootMessage;  } |

Zdrojový kód 12 Příklad přetížené metody pro přidání rozdílové hlášky, Zdroj: vlastní

9.2.3 Parsers

Tato sekce již vychází z teoretické části, kde byla popsána hlavní asynchronní operace (*CreateDbModelAsync*), která zapisuje data ze vstupních XML souborů do tříd v části *DatabaseModel.* (viz Kapitola 4.3)

Na obrázku níže je zobrazena další asynchronní operace, která se už konkrétně zabývá naplňováním databázových modelů. *reader.Name* vždy reprezentuje název elementu či atributu, na kterém se právě nachází. Díky tomuto lze snadno přidávat jeho hodnotu (*reader.Value*) do již konkrétních proměnných databázového modelu. *Reader* se na další pozici přesune pomocí příkazu *reader.ReadAsync* a jeho název i hodnota tedy převezmou vlastnosti následujícího uzlu. Ostatní asynchronní operace pak vypadají podobně a to následovně… pokud se *reader.Name* rovná požadovanému názvu, přiřaď jeho hodnotu příslušné položce v databázovém modelu a čti dál. (viz Zdrojový kód 13)

|  |
| --- |
| private static async Task AppendModel(DbModel dbModel, XmlReader reader)  {  while (await reader.ReadAsync())  {  if (reader.Name != "Model" && reader.NodeType != XmlNodeType.EndElement)  continue;  while (reader.MoveToNextAttribute())  {  if (reader.Name == "Name")  dbModel.Name = reader.Value;  else if (reader.Name == "xmlns:xsi")  dbModel.Xmlns = reader.Value;  }  await reader.ReadAsync();  while (reader.Name != "Model" || reader.NodeType != XmlNodeType.EndElement)  {  if (string.Equals(reader.Name, "SourceDriver", StringComparison.Ordinal))  await AppendSourceDriver(dbModel, reader);  else if (string.Equals(reader.Name, "DNInfo", StringComparison.Ordinal))  await AppendDnInfo(dbModel, reader);  else if (string.Equals(reader.Name, "NetBuildInfo", StringComparison.Ordinal))  await AppendNetBuildInfo(dbModel, reader);  else if (string.Equals(reader.Name, "VersionInfo", StringComparison.Ordinal))  await AppendVersionInfo(dbModel, reader);  else if (string.Equals(reader.Name, "Object", StringComparison.Ordinal))  await AppendObject(dbModel, reader);  else if (string.Equals(reader.Name, "AdditionalScripts", StringComparison.Ordinal))  await AppendAdditionalScripts(dbModel, reader);  else  await reader.ReadAsync();  }  }  }  Obrázek SEQ ObráObrázek Výsledný skript, Zdroj: vlastníObObrázek Uložení skriptu do souboru, Zdroj: vlastníní  Obrázek 5 Vzhled aplikace, Zdroj: vlastníObrázek SEQ ObráObrázek Výsledný skript, Zdroj: vlastníObObrázek Uložení skriptu do souboru, Zdroj: vlastníní |

Zdrojový kód 13 Asynchronní operace AppendModel, Zdroj: vlastní

9.2.4 Dbsc

Třída Dbsc je hlavním prvkem pro generování finálního skriptu. Obsahuje asynchronní operaci *GenerateScriptAsync*, z které jsou volány všechny ostatní potřebné metody definované v Business části. (viz Zdrojový kód 14)

Na začátku je vytvořena nová instance *StringBuilderu*, pojmenovaná *finalScript*, do které bude zapsána časová známka (*TimeStamp*) a výsledný skript. Dále se nahraje šablonový model ze streamu, který byl přijat z klientské části Blazoru. Jelikož existují tři varianty, které mohou nastat při generování skriptu (porovnání dvou databázových modelů, pouze cílový stav nebo pouze původní stav), byla vytvořena podmínka, která je odpovědná za vykonání správného bloku příkazů.

**Proces generování nové struktury databáze:**

* **Nahrání správného bloku šablony** – tato informace je zjištěna z cílového modelu a následně nahrána ze šablonového souboru.
* **Nahrání sekce *SourceDriveru*** – obsahuje informace o databázi, které jsou potřebné pro správné vygenerování SQL skriptu.
* **Vytvoření časové známky** – přidává k výslednému skriptu doplňující informace.
* **Samotné vytvoření skriptu –** vycházelo se z informací v rozdílovém modelu.

|  |
| --- |
| var finalScript = new StringBuilder();  var templateTask = templateFileParser.GetTemplateAsync(templateStream);  var template = await templateTask;  //NewDB  if (models.NewModel != null && models.PreviousModel == null)  {  var adequateTemplate = dbModelComparer.GetAdequateTemplate(null, models.NewModel, template);  var sourceDriver = GetSourceDriver(models.ModelDiff);  ScriptComposer.CreateTimeStamp(fileNamePrev, fileNameNext, adequateTemplate, sourceDriver,  finalScript);  ScriptComposer.ComposeScript(models.ModelDiff, adequateTemplate, sourceDriver,  finalScript);  }  //PrevDB  else if (models.PreviousModel != null && models.NewModel == null)  {  var adequateTemplate = dbModelComparer.GetAdequateTemplate(models.PreviousModel, null, template);  var sourceDriver = GetSourceDriver(models.ModelDiff);  ScriptComposer.CreateTimeStamp(fileNamePrev, fileNameNext, adequateTemplate, sourceDriver,  finalScript);  ScriptComposer.ComposeScript(models.ModelDiff, adequateTemplate, sourceDriver, finalScript);  }  //DbDiff  else if (models.PreviousModel != null && models.NewModel != null)  {  var adequateTemplate = dbModelComparer.GetAdequateTemplate(models.PreviousModel, models.NewModel,  template);  var sourceDriver = GetSourceDriver(models.ModelDiff);  ScriptComposer.CreateTimeStamp(fileNamePrev, fileNameNext, adequateTemplate, sourceDriver,  finalScript);  ScriptComposer.ComposeScript(models.ModelDiff, adequateTemplate, sourceDriver, finalScript);  }  return finalScript.ToString(); |

Zdrojový kód 14 GenerateScriptAsync, Zdroj: vlastní

* + 1. ScriptComposer

Poslední třída v Business části, která zajišťuje již samotné sestavení skriptu. Provádí se pomocí příkazů (*Commands*) obsažených v šablonovém souboru. Tyto příkazy obsahují již předpřipravený SQL kód, který vykonává danou operaci a parametry, které se nahrazují za konkrétní hodnoty… například příkaz *CreateTable* obsahuje kód pro vytvoření tabulky a ostatní příkazy fungují podobně. Název příkazu tedy vždy reprezentuje jeho účel.

Součástí finálního skriptu je i takzvané časové razítko (*TimeStamp*), které obsahuje dodatečné informace o vygenerovaném skriptu: (viz Zdrojový kód 15)

* **Názvy nahraných souborů**
* **Jméno výsledného souboru**
* **Čas vytvoření skriptu**
* **Prefix –** jeden z prvků v šablonovém modelu, který se přikládá ke každému vygenerovanému skriptu

|  |
| --- |
| var fileName = GetFileName(fileNamePrev, fileNameNext);  var scriptName = $"upd\_{fileName}.sql";  var now = DateTime.Now;  var formatedDate = now.ToString("yyyyMMdd hh:mm:ss");  finalScript.Append($"-- Jméno souboru skriptu: {fileName}\n");  finalScript.Append($"-- Výchozí datový model: {fileNamePrev}\n");  finalScript.Append($"-- Nový datový model: {fileNameNext}\n");  finalScript.Append($"-- Časové razítko: {now}\n\n");  if (fileNamePrev!=null)  finalScript.Append(string.Format(adequateTemplate.Commands["UpgradePrefix"],  sourceDriver.Database, scriptName, formatedDate));  else  finalScript.Append(string.Format(adequateTemplate.Commands["NewDBPrefix"],  sourceDriver.Database, scriptName, formatedDate)); |

Zdrojový kód 15 Časové razítko, Zdroj: vlastní

Metoda *ComposeScript* již pracuje přímo s danou šablonou (*adequateTemplate*) a podle ní potom generuje výsledný skript. První se do skriptu přidají modifikace týkající se tabulek a následně *AdditionalScripts*, což jsou uživatelské skripty, které si mění uživatel vzhledem k používané databázi. (viz Zdrojový kód 16)

|  |
| --- |
| var tablePrefix = adequateTemplate.Commands["TablePrefix"];  var viewPrefix = adequateTemplate.Commands["FullViewPrefix"];  ChangedObjects(modelDiff, adequateTemplate, finalScript, tablePrefix);  NewObjects(modelDiff, adequateTemplate, finalScript, sourceDriver.Username,  tablePrefix);  RemovedObjects(modelDiff, adequateTemplate, finalScript, sourceDriver.Username,  tablePrefix, viewPrefix);  finalScript.Append("\n\n/\* \_\_\_\_\_\_\_\_\_\_ USER SCRIPT \_\_\_\_\_\_\_\_\_\_ \*/");  AdditionalScripts(modelDiff, adequateTemplate, finalScript); |

Zdrojový kód 16 Metoda ComposeScript, Zdroj: vlastní

Na následujícím zdrojovém kódu je detailní pohled na metodu *NewObjects*, jejíž výsledkem je SQL kód pro vygenerování nové tabulky i s jejími atributy.

Prvně se do proměnných načtou požadované příkazya následně cyklus prochází všechny objekty, které byly přidány do rozdílového modelu jako *NewObject*. Z těchto objektů se v dalším kroku načtou jeho atributy a hodnoty se pomocí metody *GenerateTableRIV* jen předají jako parametr do příslušného příkazu.

Přidaní příkazu k výslednému skriptu je řešeno pomocí metody *Append*, kterou nabízí přímo třída *StringBuilderu*. Předává se jí zformátovaný textový řetězec, jehož první parametr je šablonový příkaz, ty další poté reprezentují jeho konkrétní parametry.

|  |
| --- |
| var createTable = adequateTemplate.Commands["CreateTable"];  var addCv = adequateTemplate.Commands["AddColumnValues"];  adequateTemplate.Commands.TryGetValue("BeforeCreateTable", out var bCT);  foreach (var news in modelDiff.NewObjects.Values)  {  var allAttrs = new SortedList<string,string>();  var allValues = new SortedList<string,string>();  var allrows = new SortedList<string,string>();  var tpf = GetTablePrefix(news.Base, tablePrefix);  GenerateTableRIV(news, news.DbAttributes.Values, allAttrs, allrows, adequateTemplate,  allValues);    var rows = $"\t{string.Join(",\n\t", allrows.Values)},";  var requiredAttrs = string.Join(", ", allAttrs.Values);  var requiredValues = string.Join(", ", allValues.Values);  var tableName= string.Format(tpf,news.SourceName);  finalScript.Append(string.Format($"\n\n{bCT}", $"{tableName}"));  finalScript.Append(string.Format(createTable, $"{tableName}", userName, "ID",  $"{rows}\n"));  finalScript.Append(string.Format(addCv, $"{tableName}", requiredAttrs,  requiredValues));  } |

Zdrojový kód 17 Metoda NewObjects, Zdroj: vlastní

Například příkaz *CreateTable*, uvedený níže, používá čtyři parametry (*%0:s*, *%1:s,* *%2:s* a *%3:s*). Každý potom představuje určitou část v tomto SQL kódu, která se liší v závislosti na aktuálně zpracovávané tabulce.

**Parametry se tedy zaměňují za následující hodnoty:**

* **%0:s** – Název tabulky
* **%1:s** – Název databáze
* **%2:s** – ID
* **%3:s** – Řádky databáze (atributy)

|  |
| --- |
| <command name="CreateTable">PRINT 'Vytvoreni tabulky %0:s'  GO  CREATE TABLE [%1:s].[%0:s] (  %3:sCONSTRAINT [%0:s\_PK] PRIMARY KEY CLUSTERED ([%2:s]))  GO  </command> |

Zdrojový kód 18 Příkaz CreateTable, Zdroj: vlastní

**Výsledek, který je potom přidán k výslednému skriptu může vypadat takto:**

|  |
| --- |
| PRINT 'Vytvoreni tabulky T$test\_test'  GO  CREATE TABLE [testDatabaze].[T$test\_test] (  [classname] varchar(64),  [ID] bigint IDENTITY(1, 1) NOT NULL,  CONSTRAINT [T$test\_test\_PK] PRIMARY KEY CLUSTERED ([ID]))  GO  INSERT INTO #OWNER#.[T$test\_test] (ID)  VALUES(0)  GO |

Zdrojový kód 19 Výsledek zapsaný ve finálním skriptu, Zdroj: vlastní

9.3 Tests

Jedná se o sekci vyhrazenou speciálně pro testování jednotlivých částí aplikace. Obsahuje třídu s konkrétními testy (*ScriptGenerationTests*) a testovací soubory (*Resources*), které jsou datové modely, šablona a DTD soubor.

Testy jsou nedílnou součástí celého projektu, protože pomáhají šetřit čas při ověřování správného fungování konkrétních částí v kódu. Hlavní výhodou je, že se nemusí spouštět a sestavovat celá aplikace. V pozdějších fázích vývoje, kdy je potřeba otestovat nějakou konkrétní funkci, která se vykonává až k samotnému závěru, by totiž bylo nutné projít několika kroky, které by předcházely jejímu spuštění… takto pouze stačí zadat vstupní data do testu naprogramovaného pro onu funkci a následně jej spustit.

**Pro označení testů byly použity dva atributy:**

* **Theory** – označí následující testovanou metodu jako datovou teorii. Datové teorie jsou chápány jako testy, které přijímají data z datového zdroje.
* **InlineData** – představuje datový zdroj pro datové teorie, jinými slovy určuje hodnoty pro tyto vstupy

Ve finální fázi vývoje aplikace byla využívána testovací metoda pro vygenerování finálního skriptu. Tato metoda byla označena výše uvedenými atributy a zdrojová data pak předána jako parametry.

Jelikož se jedná o jednu z posledních operací, jenž se v aplikaci provádí, bylo nutné zavolat všechny předchozí metody, které jsou nezbytné pro její správný průchod. Bylo tedy potřeba na začátek vytvořit instance *Parserů, Compareru* a *StringBuilderu*. Přes tyto instance byly dále volány metody v nich obsažené, které slouží k vytvoření datových modelů, rozdílového modelu a nahrání správné šablony. Až poté je možné zavolat požadovanou funkci.

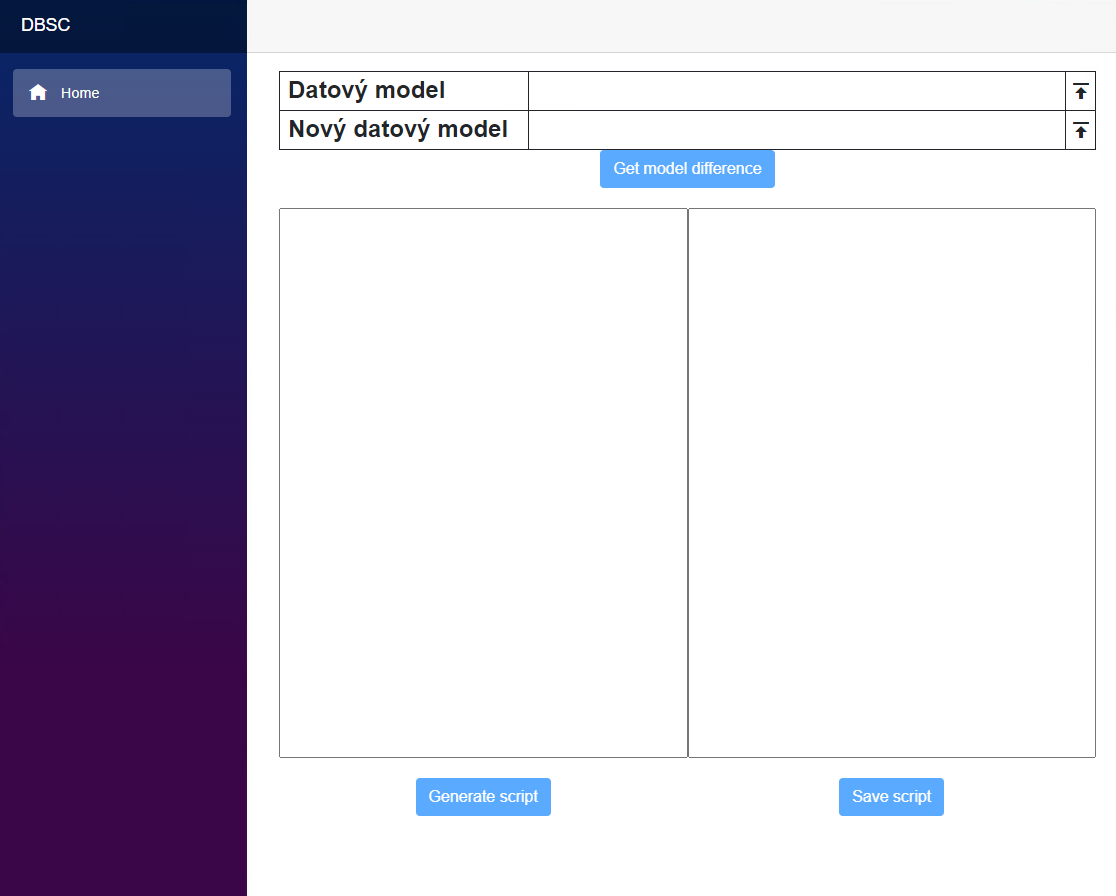
Jak již bylo zmíněno v úvodu této kapitoly, takový test potom ušetří při vývoji velké množství času, protože nemusíme sestavovat celou aplikaci a zbytečně v ní procházet jednotlivými kroky. Zde totiž není kromě definování zdrojových dat při psaní testu potřeba žádná další akce od programátora.

Další výhodou je také možnost rychlého krokování v kódu. Je možné si totiž na libovolné místo dát takzvaný *BreakPoint*, díky kterému se spouštění aplikace pozastaví v tomto bodě. Pomocí klávesové zkratky F10 nebo F11 potom lze postupně procházet kroky, které aplikace vykonává, a je tak umožněno nahlédnout, v jakém konkrétním místě se stala chyba.

|  |
| --- |
| [Theory]  [InlineData(@"Resources\JadroTest\_DBSC\_puvodni.xml",  @"Resources\JadroTest\_DBSC\_novy.xml", @"Resources\templatesDN.xml")]  public async void ComposeScript(string filePath, string filePath2, string templatePath)  {  var dbModelParser = new DbModelParser();  var dbModelComparer = new DbModelComparer();  var dbFileParser = new TemplateFileParser();  var finalScript = new StringBuilder();  var dbTemplate = await dbFileParser.GetTemplateAsyncFromFile(templatePath);  var prevDbModel = await dbModelParser.CreateDbModelAsyncFromFiles(filePath);  var nextDbModel = await dbModelParser.CreateDbModelAsyncFromFiles(filePath2);  var diff = dbModelComparer.CompareModel(prevDbModel, nextDbModel);  var adequateTemplate = dbModelComparer.GetAdequateTemplate(prevDbModel,  nextDbModel, dbTemplate);  var sourceDriver = DBSC.Core.Dbsc.GetSourceDriver(diff);  ScriptComposer.CreateTimeStamp(filePath, filePath2, adequateTemplate,  sourceDriver, finalScript);  ScriptComposer.ComposeScript(diff, adequateTemplate, sourceDriver,  finalScript);  var s = finalScript.ToString();  } |

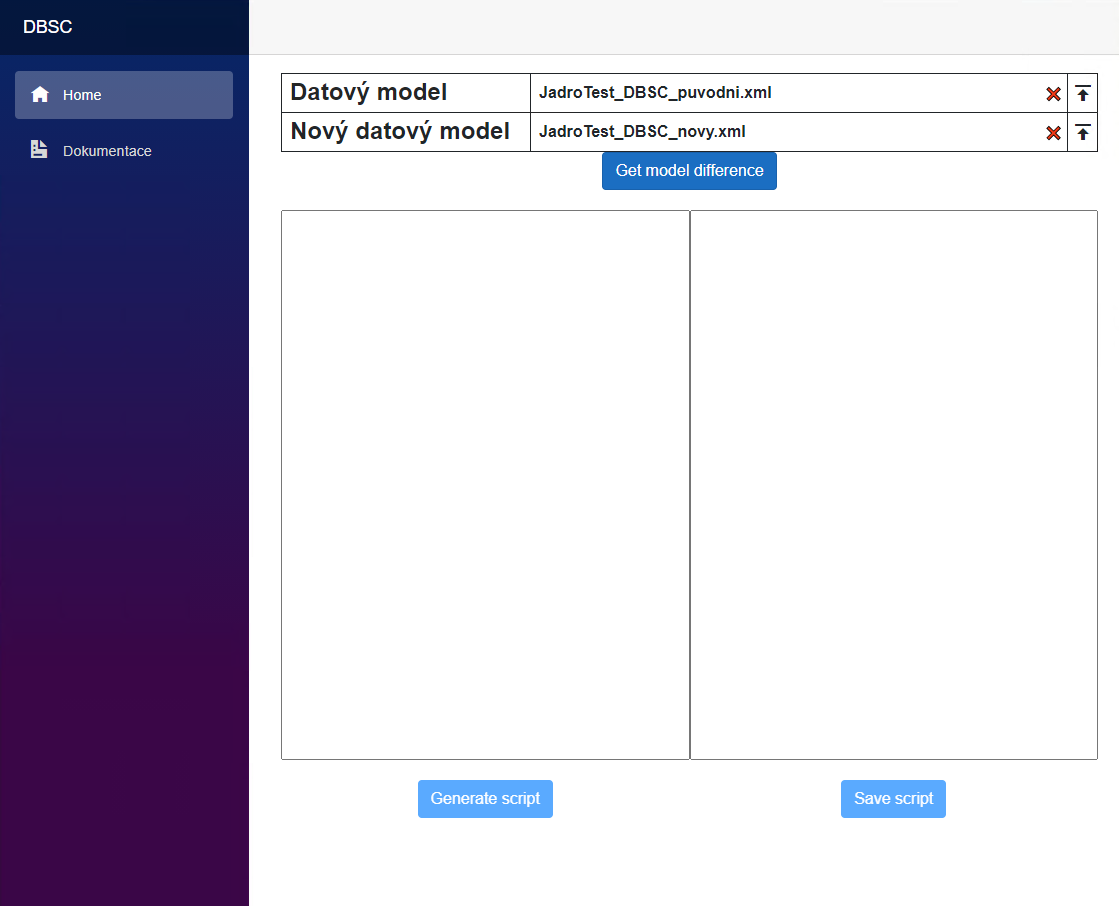
Zdrojový kód 20 Ukázka testu na vytvoření finálního skriptu, Zdroj: vlastní

1. Finální aplikace

Při načtení aplikace jsou přístupná pouze tlačítka pro nahrání vstupních souborů. Ostatní tlačítka jsou prozatím „vypnuta“. (viz Obrázek 5)

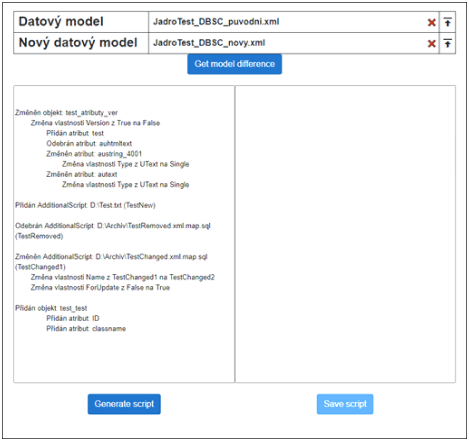
Obrázek 5 Vzhled aplikace, Zdroj: vlastní

Po nahrání souboru reprezentujícího cílový stav databáze, případně po nahrání i původního stavu, se tlačítko zpřístupní a je tedy možné pokračovat dál. (viz Obrázek 6)



Obrázek 6 Nahrání souborů, Zdroj: vlastní

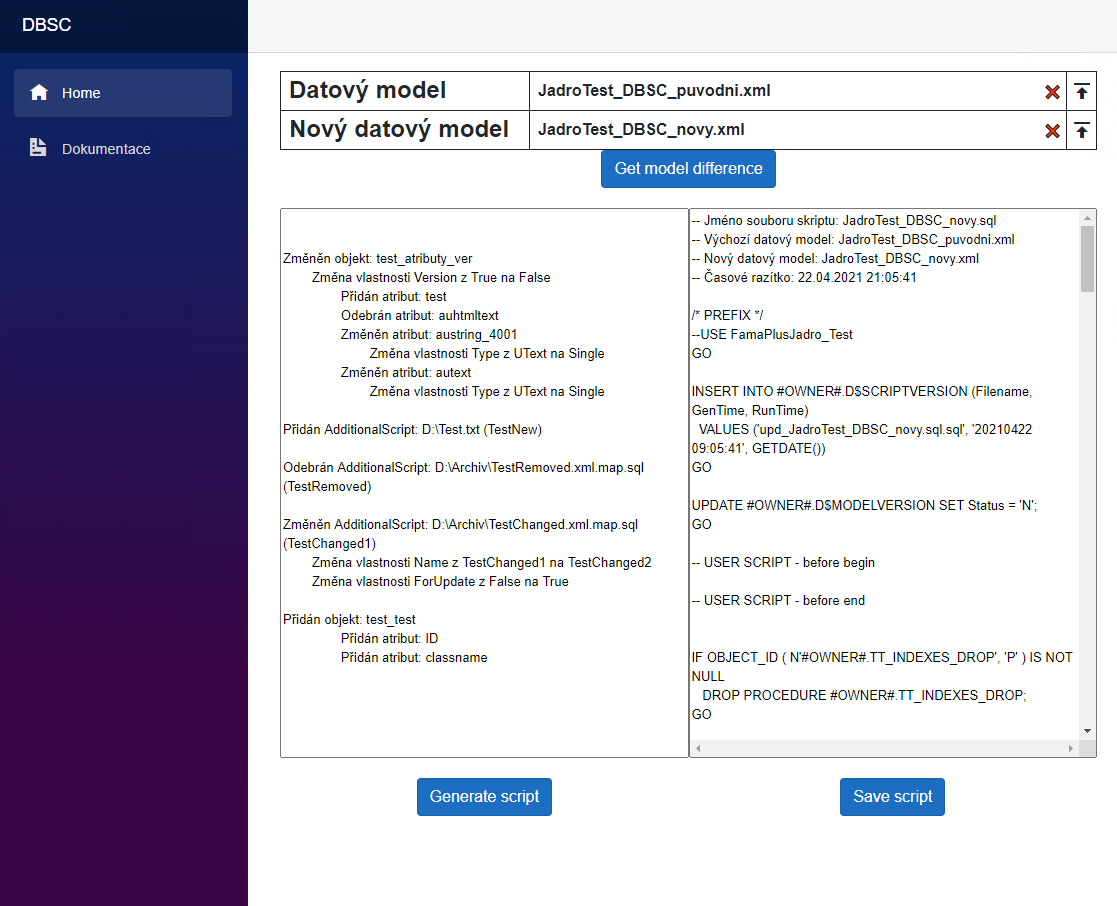
Po stisknutí tlačítka *„Get model difference“* se v levé textové oblasti zobrazí rozdílové hlášky, které byly zaznamenány při vytváření rozdílového modelu. Na základě těchto informací se může uživatel rozhodnout, zda se jedná o zásadní změny v databázi, a je tedy nutné vygenerovat celý skript, nebo stačí jen pár drobných úprav, které uživatel dokáže provést sám. (viz Obrázek 7)



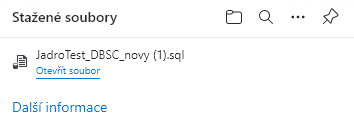
Obrázek 7 Rozdílové hlášky, Zdroj: vlastní

Pokud se uživatel rozhodne pokračovat v aplikaci, zpřístupní se mu tlačítko *„Generate script“*, jež následně tento skript vytvoří pomocí rozdílového modelu. (viz Obrázek 8)

Výsledný skript se v dalším kroku vypíše v pravé textové oblasti a zároveň se i zpřístupní tlačítko *„Save script“*, které vypsaný skript uloží do samostatného souboru. (viz Obrázek 9)



Obrázek 8 Výsledný skript, Zdroj: vlastní



Obrázek 9 Uložení skriptu do souboru, Zdroj: vlastní

1. Publikování aplikace

Aby mohli uživatelé volně přistupovat k aplikaci je nutné ji nejprve publikovat. Toho lze dosáhnout hned několika způsoby, které Visual Studio nabízí:

* **Azure** – publikování na cloudové úložiště společnosti Microsoft
* **Registr kontejneru Dockeru** – pokud aplikace používá Docker
* **Složka** – aplikace se publikuje do složky
* **Server FTP/FTPS** – umožní publikovat aplikaci na jiný server, než je Azure
* **Webový server (IIS)** – publikování na server v rámci Internetové Informační Služby
* **Importovat profil** – umožňuje importování profilu při publikování na IIS nebo Azure



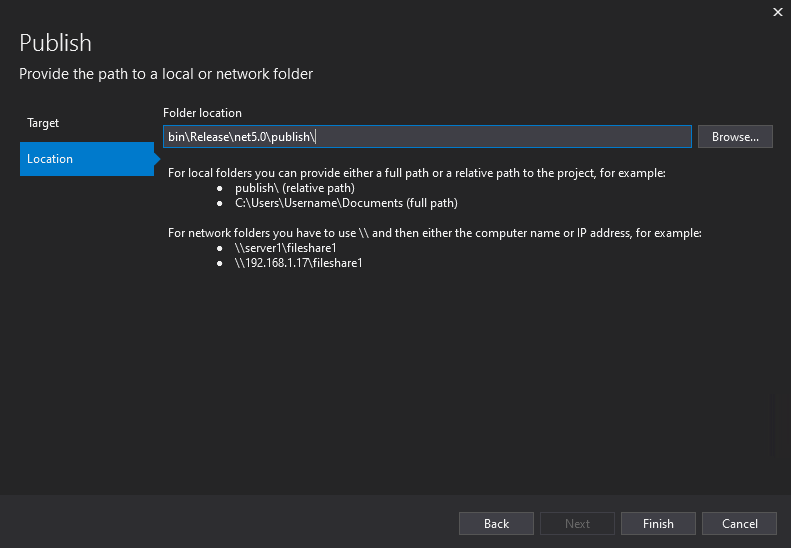
Obrázek 10 Publikování – možnosti, Zdroj: vlastní

11.1 Publikování aplikace do složky

Pro názornou demonstraci fungování aplikace je na přiložené CD nahrána finální verze programu za pomoci této možnosti.

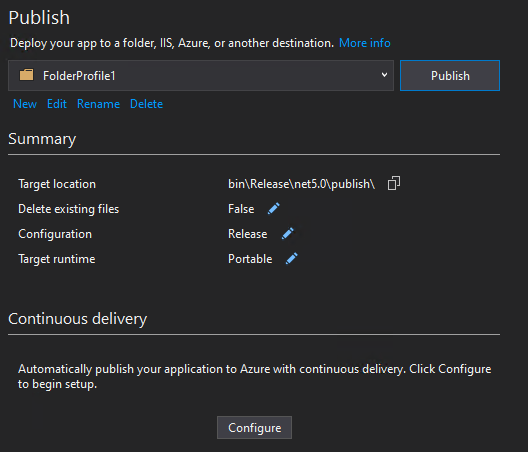
V rámci této možnosti jsou soubory aplikace zkopírovány do konkrétní složky v úložišti počítače. Publikování do složky nejčastěji slouží k testování, jelikož se jedná o rychlý a jednoduchý způsob. Dále může sloužit i jako sdílený soubor, který pak lze nasadit do různých serverů, například přes správce internetových služeb.

Pomocí pravého tlačítka myši stačí kliknout na požadovaný projekt a zvolit *Publish*. Po tomto kroku se zobrazí možnosti, kde se zvolí *Folder* (viz Obrázek 10). Dále je nutné vybrat umístění, kde bude publikovaná aplikace uložena. V základu je tato cesta uvedena v projektové složce (viz Obrázek 11).



Obrázek 11 Publikování – cesta, Zdroj: vlastní

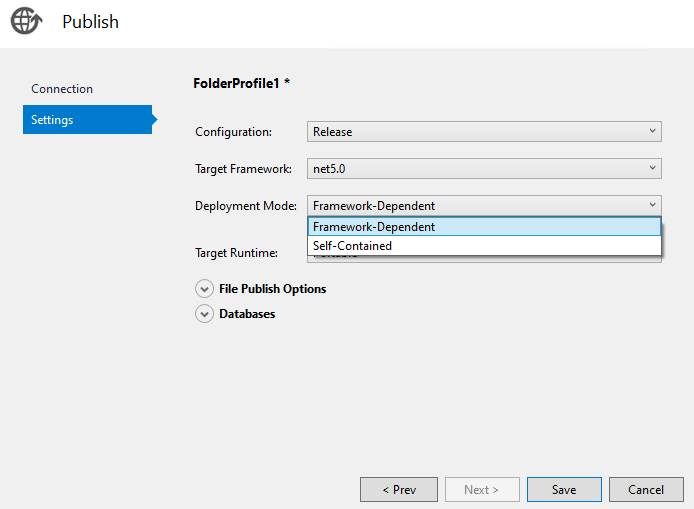
Po zvolení cílového umístění se vytvoří profil, který obsahuje nastavení pro samotné publikování. (viz Obrázek 12)



Obrázek 12 Publikování – profil, Zdroj: vlastní

Dle potřeby lze tato nastavení ještě dále upravovat. Při kliknutí na tlačítko *Edit*, které je umístěno pod názvem profilu, se zobrazí panel, kde je možnost tyto dodatečné modifikace provést. Hlavním bodem je *Deployment Mode* neboli *mód* *nasazení*. Zde je možné vybrat ze dvou variant:

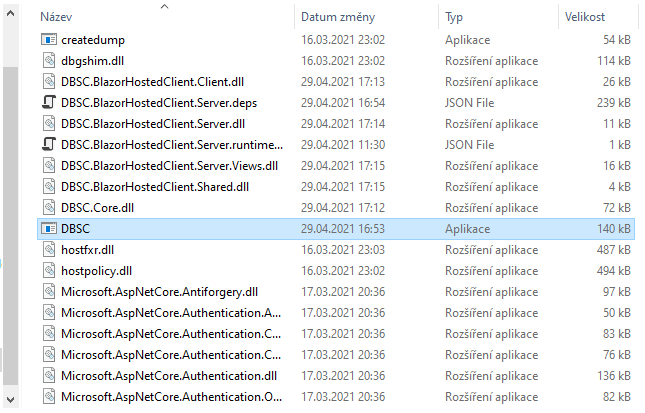
* **Framework-Dependent** – spustitelný soubor je multi-platformní, to znamená, že ji lze spustit na různých počítačových platformách (například Linux nebo Windows), ale neobsahuje *.NET runtime.* Je tedy nutné, aby si jej uživatel nainstaloval.
* **Self-Contained** – takto publikovaná aplikace cílý na specifickou platformu. Tou může být například již zmiňovaný Windows. Výsledná složka je však značně větší než u varianty *Framework-dependent*, protože již obsahuje *.NET runtime*.



Obrázek 13 Publikování – výběr módu nasazení, Zdroj: vlastní

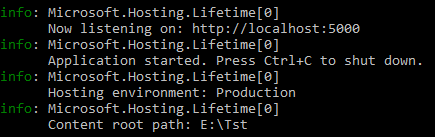
Ve výsledné složce jsou potom umístěny všechny potřebné soubory, které aplikace potřebuje ke správnému chodu. Pomocí spustitelného souboru, jež je v této složce také obsažen, se spustí hosting na lokálním počítači (localhost).

Všechny výše uvedené kroky byly již v rámci pulikování provedy, a na přiloženém CD se pak jen nachází složka obsahující soubory aplikace. Pomocí módu *Self-Contained* byla cílena na platformu Windows x64. Není tedy nutné dodatečně instalovat *.NET runtime*, ale můžete aplikaci v rámci Windows rovnou spustit.

Při spuštění aplikace *DBSC,* která je hlavním spustitelným souborem, se zobrazí konzolová aplikace obsahující následující informace:

Obrázek 14 Publikování – výsledná složka, Zdroj: vlastní

* Adresu, na které je možné aplikaci spustit
* Informaci o spuštění aplikace
* Název hostujícího prostředí
* Umístění aplikace



Obrázek 15 Publikování – spustitelný soubor, Zdroj: vlastní

Aplikace potom dále běží, dokud není tento soubor vypnut, čehož lze docílit pomocí klávesové kratky *Ctrl+C* nebo prostým kliknutím na křížek v pravém horním rohu.

Je důležité zmínit, že aplikace byla vyvýjena tak, aby správně fungovala v prohlížeči Google Chrome nebo Microsoft Edge, v ostatních prohlížečích by se kaskádové styly nemuseli správně načíst. Možné komplikace mohou nastat i při spuštění apliakce z CD. Doporučuji si tedy aplikaci nejprve přesunout na lokální úložiště.

11.2 Obfuskace aplikace

Aby bylo co nejtěžší dostat zdrojové kódy z aplikace, byly soubory, nahrané na přiloženém CD, obfuskovány pomocí programu Skater Light Edition.

Obfuskace je vlastně proces šifrování dat. Pomocí algoritmů převádí zdrojový kód v konkrétním programovacím jazyce do jeho kopie, ale provede v něm několik následujících změn, aby jej nešlo tak snadno rozluštit:

* **Odstranění komentářů**
* **Odstranění formátování**
* **Přejmenování proměnných, konstant i metod**
* **Vložením „mrtvého“ nebo duplicitního kódu**

Na trhu je dostupná celá řada různých obfuskátorů, které nabízejí i doplňující funkce. Liší se pak složitostí prováděných algoritmů a cenou. Ačkoli existuje i několik obfuskátorů, které jsou nabízeny zdarma pro širokou veřejnost, je stále více společností, které si za tuto službu nechávají zaplatit. Najít jednoduchý a účinný obfuskátor, který by byl zdarma je tedy stále těžší.

|  |
| --- |
| var tablePrefix = adequateTemplate.Commands["TablePrefix"];  var viewPrefix = adequateTemplate.Commands["FullViewPrefix"];  ChangedObjects(modelDiff, adequateTemplate, finalScript, tablePrefix);  NewObjects(modelDiff, adequateTemplate, finalScript, sourceDriver.Username,  tablePrefix);  RemovedObjects(modelDiff, adequateTemplate, finalScript, sourceDriver.Username,  tablePrefix, viewPrefix);  finalScript.Append("\n\n/\* \_\_\_\_\_\_\_\_\_\_ USER SCRIPT \_\_\_\_\_\_\_\_\_\_ \*/");  AdditionalScripts(modelDiff, adequateTemplate, finalScript); |

Zdrojový kód 21 Originální kód, Zdroj: vlastní

|  |
| --- |
| string command1 = adequateTemplate.Commands["TablePrefix"];  string command2 = adequateTemplate.Commands["FullViewPrefix"];  ScriptComposer.feeee(modelDiff, adequateTemplate, finalScript, command1);  ScriptComposer.geeeg(modelDiff, adequateTemplate, finalScript,  sourceDriver.Username, command1);  ScriptComposer.\u1737(modelDiff, adequateTemplate, finalScript,  sourceDriver.Username, command1, command2);  finalScript.Append("\n\n/\* \_\_\_\_\_\_\_\_\_\_ USER SCRIPT \_\_\_\_\_\_\_\_\_\_ \*/");  ScriptComposer.leeee(modelDiff, adequateTemplate, finalScript); |

Zdrojový kód 22 Kód po obfuskaci, Zdroj: vlastní

Závěr

Výsledkem této práce je webová aplikace, která slouží pracovníkům firmy Tesco SW ke generování struktur databází. Aplikace umožňuje výběr jednoho až dvou souborů, které jsou následně navzájem porovnány pro vytvoření rozdílových hlášek a modelu, který slouží k vygenerování výsledného skriptu. Tento skript si uživatel může z textové oblasti zkopírovat nebo pomocí příslušného tlačítka stáhnout.

Uživatelské rozhraní je řešeno za použití moderní technologie Blazor vyvíjené společností Microsoft. Samotné programování logiky je pak v jazyce C#.

Hlavním cílem této práce bylo nejen vytvoření funkční aplikace, jak již bylo zmíněno v úvodu, ale i usnadnění práce zaměstnancům firmy Tesco SW, kteří s databázemi pracují. Přidáním mezikroku, který zobrazuje rozdílové hlášky v textové oblasti, si uživatel může ušetřit spoustu času, jelikož nemusí generovat celý skript, který může mít až okolo 65 MB. Dalším cílem bylo vytvoření přehledného a lehce ovladatelného uživatelského rozhraní. Zde se narozdíl od původní aplikace odstranila nutnost použití checkboxu, který reprezentoval rozhodnutí, zda chce uživatel databáze porovnávat nebo generovat. Jeho použití bylo v aplikaci velmi nepřehledné a raději se nahradilo vnitřní logikou, kdy aplikace sama rozhodne, zda pracuje sedvěma soubory (porovnává dva stavy databáze) nebo pouze s jedním souborem (tvoří novou strukturu DB).

Do budoucna bych chtěl aplikaci dál rozvíjet a vytvořit například konzolovou verzi aplikace, která by využívala stejné logiky, jako současná aplikace, avšak sloužila by zkušenějším uživatelům, kteří nevyžadují možnost mezikroku s vypsáním rozdílových hlášek.

POUŽITÁ LITERATURA

HERNANDEZ, Michael James. Návrh databází. Přeložil Jan BOUDA. Praha: Grada Publishing, 2006. Profesionál. ISBN 80-247-0900-7

KOSEK, Jiří. PHP a XML. Praha: Grada, 2009. Profesionál. ISBN 978-80-247-1116-4

OPPEL, Andrew J. Databáze bez předchozích znalostí. Přeložil David KRÁSENSKÝ. Brno: Computer Press, 2006. ISBN 80-251-1199-7

ROBINSON, Simon. C#: programujeme profesionálně. Přeložil Bogdan KISZKA. Brno: Computer Press, 2003. ISBN 80-251-0085-5.

OSTATNÍ ZDROJE

**Blazor**

Goodbye JavaScript, Hello WebAssembly. Telerik & Kendo UI – .NET Components Suites & JavaScript UI Libraries [online]. Copyright © 2021, Progress Software Corporation and [cit. 15.04.2021]. Dostupné z: <https://www.telerik.com/blogs/goodbye-javascript-hello-webassembly>

Single-page application vs. multiple-page application | by Neoteric | Medium. Medium – Where good ideas find you. [online]. Dostupné z: <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>

What is Blazor? – Blazor University. Blazor University – Learn the new .NET SPA framework from Microsoft [online]. Copyright © 2019 [cit. 15.04.2021]. Dostupné z: <https://blazor-university.com/overview/what-is-blazor/>

What's the Difference Between Blazor vs Razor. Telerik & Kendo UI – .NET Components Suites & JavaScript UI Libraries [online]. Copyright © 2021, Progress Software Corporation and [cit. 15.04.2021]. Dostupné z: <https://www.telerik.com/blogs/difference-between-blazor-vs-razor>

**C#**

Handling Exceptions in C# – When to catch them, where to catch them, and how to catch them - YouTube. YouTube [online]. Copyright © 2021 Google LLC [cit. 24.04.2021]. Dostupné z: <https://www.youtube.com/watch?v=LSkbnpjCEkk&list=PLLWMQd6PeGY2GVsQZ-u3DPXqwwKW8MkiP&index=8>

Working With The File System in C# – Managing Folders and Files – YouTube. YouTube [online]. Copyright © 2021 Google LLC [cit. 24.04.2021]. Dostupné z: <https://www.youtube.com/watch?v=9mUuJIKq40M&list=PLLWMQd6PeGY2GVsQZ-u3DPXqwwKW8MkiP&index=14>

XmlSerializer.Deserialize Metoda (System.Xml.Serialization) | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 24.04.2021]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/api/system.xml.serialization.xmlserializer.deserialize?view=net-5.0>

**Databáze**

Co je databáze. Jan Štráfelda: průvodce online projektem [online]. [cit. 24.04.2021]. Dostupné z: <https://www.strafelda.cz/databaze>

Co je to databáze | Oracle Česká republika. [online]. [cit. 15.04.2021] Dostupné z: <https://www.oracle.com/cz/database/what-is-database/>

Databáze. Databáze [online]. Copyright © 2010 Misha [cit. 24.04.2021]. Dostupné z: <http://www.databaze.chytrak.cz/modely.htm>

Oracle vs MSSQL | 22 Important Comparison You Should Know. Best Online Training & Video Courses | eduCBA [online]. Copyright © 2020 [cit. 24.04.2021]. Dostupné z: <https://www.educba.com/oracle-vs-mssql/>

OTTE, L. (2013): Databázové systémy. Vysoká škola báňská, Ostrava, 189 s. ISBN 978–80–248–3054–4. [cit. 24.04.2021]. Dostupné z: <http://projekty.fs.vsb.cz/463/edubase/VY_01_044/Databázové%20systémy.pdf>

**Electron**

Build a Desktop App with Electron... But Should You? - YouTube. YouTube [online]. Copyright © 2021 Google LLC [cit. 24.04.2021]. Dostupné z: <https://www.youtube.com/watch?v=3yqDxhR2XxE>

Electron, 1. část - Kutáč.cz . Portfolio Kutáč.cz [online]. Copyright © [cit. 24.04.2021]. Dostupné z: <https://www.kutac.cz/weby-a-vse-okolo/electron-1-cast>

**XML**

Computerworld.cz | Deník pro IT profesionály [online]. [cit. 15.04.2021]. Dostupné z: <https://computerworld.cz/archiv/co-je-to-parser-11805>

C# XmlReader – reading XML in C# with XmlReader. ZetCode, tutorials for programmers [online]. Copyright © 2007 [cit. 15.04.2021]. Dostupné z: <https://zetcode.com/csharp/xmlreader/>

XML Processing Options | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 15.04.2021]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/standard/data/xml/xml-processing-options>

**Publikování**

Deploy to a local folder – Visual Studio | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 24.04.2021]. Dostupné z: <https://docs.microsoft.com/en-us/visualstudio/deployment/quickstart-deploy-to-local-folder?view=vs-2019>

First look at deployment – Visual Studio | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 24.04.2021]. Dostupné z: <https://docs.microsoft.com/en-us/visualstudio/deployment/deploying-applications-services-and-components?view=vs-2019>

SEZNAM ZKRATEK

**CSS** Cascading Style Sheets (Kaskádový styl)

**DB** Database (Databáze)

**DBMS** Database management system

**DBSC** Database Script Creator

**DCL** Data Control Language

**DDL** Data Definition Language

**DML** Data Manipulation Language

**DTD** Document Type Definition (Definice typu dokumentu)

**DQL** Data Query Language

**FTP** File Transfer Protocol

**FTPS** File Transfer Protocol Secure

**HTML** Hypertext Markup Language

**ID** Identification (index)

**IIS** Internet Information Service

**MB** Megabyte

**MSSQL** Microsoft SQL Server

**OS** Operační systém

**SŘBD** Systém řízení báze dat

**SQL** Structured Query Language

**UI** User Interface (Uživatelské rozhraní)

**XMI** XML Metadata Interchange

**XML** Extensible Markup Language

SEZNAM OBRÁZKŮ

[Obrázek 1 Původní DBSC, Zdroj: vlastní 8](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622405)

[Obrázek 2 Konzolová aplikace, Zdroj: vlastní 10](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622406)

[Obrázek 3 XML vysvětlivky, Zdroj: vlastní 12](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622407)

[Obrázek 4 Struktura projektu, Zdroj: vlastní 24](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622408)

[Obrázek 5 Vzhled aplikace, Zdroj: vlastní 36](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622410)

[Obrázek 6 Nahrání souborů, Zdroj: vlastní 37](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622411)

[Obrázek 7 Rozdílové hlášky, Zdroj: vlastní 37](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622412)

[Obrázek 8 Výsledný skript, Zdroj: vlastní 38](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622413)

[Obrázek 9 Uložení skriptu do souboru, Zdroj: vlastní 38](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622414)

[Obrázek 10 Publikování – možnosti, Zdroj: vlastní 39](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622415)

[Obrázek 11 Publikování – cesta, Zdroj: vlastní 40](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622416)

[Obrázek 12 Publikování – profil, Zdroj: vlastní 40](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622417)

[Obrázek 13 Publikování – výběr módu nasazení, Zdroj: vlastní 41](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622418)

[Obrázek 14 Publikování – výsledná složka, Zdroj: vlastní 42](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622419)

[Obrázek 15 Publikování – spustitelný soubor, Zdroj: vlastní 42](file:///C:\Users\Jaroslav%20Sýkora\Downloads\BP_Sýkora_finalni_verze.docx#_Toc70622420)

SEZNAM ZDROJOVÝCH KÓDŮ

[Zdrojový kód 1 Deserializace XML, Zdroj: vlastní 14](#_Toc70680167)

[Zdrojový kód 2 Ukázka kódu Blazor, Zdroj: vlastní 17](#_Toc70680168)

[Zdrojový kód 3 Ukázka části code v Blazoru, Zdroj: vlastní 17](#_Toc70680169)

[Zdrojový kód 4 Ukázka kódu SQL, Zdroj: vlastní 20](#_Toc70680170)

[Zdrojový kód 5 Ořezaná ukázka vstupního XML, Zdroj: vlastní 23](#_Toc70680171)

[Zdrojový kód 6 Funkce při spuštění aplikace, Zdroj: vlastní 25](#_Toc70680172)

[Zdrojový kód 7 Metoda pro uložení skriptu, Zdroj: vlastní 25](#_Toc70680173)

[Zdrojový kód 8 Volání JavaScriptové metody, Zdroj: vlastní 26](#_Toc70680174)

[Zdrojový kód 9 Metoda pro vytvoření rozdílového souboru a hlášek, Zdroj: vlastní 27](#_Toc70680175)

[Zdrojový kód 10 Řadič, Zdroj: vlastní 28](#_Toc70680176)

[Zdrojový kód 11 Metoda CompareModel, Zdroj: vlastní 29](#_Toc70680177)

[Zdrojový kód 12 Příklad přetížené metody pro přidání rozdílové hlášky, Zdroj: vlastní 29](#_Toc70680178)

[Zdrojový kód 13 Asynchronní operace AppendModel, Zdroj: vlastní 30](#_Toc70680179)

[Zdrojový kód 14 GenerateScriptAsync, Zdroj: vlastní 31](#_Toc70680180)

[Zdrojový kód 15 Časové razítko, Zdroj: vlastní 32](#_Toc70680181)

[Zdrojový kód 16 Metoda ComposeScript, Zdroj: vlastní 33](#_Toc70680182)

[Zdrojový kód 17 Metoda NewObjects, Zdroj: vlastní 33](#_Toc70680183)

[Zdrojový kód 18 Příkaz CreateTable, Zdroj: vlastní 34](#_Toc70680184)

[Zdrojový kód 19 Výsledek zapsaný ve finálním skriptu, Zdroj: vlastní 34](#_Toc70680185)

[Zdrojový kód 20 Ukázka testu na vytvoření finálního skriptu, Zdroj: vlastní 36](#_Toc70680186)

[Zdrojový kód 21 Originální kód, Zdroj: vlastní 43](#_Toc70680187)

[Zdrojový kód 22 Kód po obfuskaci, Zdroj: vlastní 43](#_Toc70680188)

ANOTACE

**Bibliografický údaj:** Sýkora, Jaroslav. Re-engineering nástroje DBSC a procesu generování databáze. Olomouc 2021. Bakalářská práce. Moravská vysoká škola Olomouc. Vedoucí práce: Ing. Jiří Blahuta, Ph.D.

**Název práce:** Re-engineering nástroje DBSC a procesu generování databáze

**Autor:** Jaroslav Sýkora

**Ústav:** Ústav informatiky a aplikované matematiky

**Vedoucí práce:** Ing. Jiří Blahuta, Ph.D.

**Abstrakt:** Bakalářská práce je zaměřena na re-engineering aplikace DBSC, která zaměstnancům firmy TescoSW umožňuje generovat skripty pro aktualizaci databáze nebo k vytvoření její nové struktury. V teoretické části jsou v jednotlivých kapitolách popsány technologie použité při vývoji a další pojmy, které jsou úzce spojeny s tímto procesem. Poznatky z těchto kapitol jsou pak využívány v praktické části, která je zaměřena na samotné řešení aplikace. Dále je zde popsán proces reverzního inženýrství a představení nové aplikace společně s popsáním jednotlivých kroků, které musí uživatel provést, aby dosáhl požadovaného výsledku.

**Klíčová slova:** C#, webová aplikace, databáze, reverzní inženýrství, Blazor, uživatelské rozhraní, XML

ANOTATION

**Bibliographic detail:** Sykora, Jaroslav. Re-Engineering of DBSC tool and process of database generating. Olomouc 2021. Bachelor thesis. Moravian Business College Olomouc. Thesis supervisor: Ing. Jiří Blahuta, Ph.D.

**Title:** Re-Engineering of DBSC tool and process of database generating

**Author:** Jaroslav Sykora

**Department:** Department of informatics and applied mathematics

**Thesis supervisor:** Ing. Jiri Blahuta, Ph.D.

**Abstract:** This Bachelor thesis focuses on re-engineering of DBSC application, which allows Tesco SW employees to generate scripts for database update or to create its new structure. In theoretical part are described technologies that were used for developement and other concepts, which are closely related to this process. Findings from theoretical part are then used in practical part, which is focused to the application developement itself. Then there is described process of reverse engineering and presentation of new application together with description of each step a user has to do to acomplish the desired result.

**Keywords:** C#, web application, database, reverse engineering, Blazor, user interface, XML

1. 301 Moved Permanently [online]. Copyright ©D [cit. 09.04.2021]. Dostupné z: <http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text.pdf> [↑](#footnote-ref-1)
2. C# XmlReader - reading XML in C# with XmlReader. ZetCode, tutorials for programmers [online]. Copyright © 2007 [cit. 10.04.2021]. Dostupné z: <https://zetcode.com/csharp/xmlreader/> [↑](#footnote-ref-2)
3. Single-page application vs. multiple-page application | by Neoteric | Medium. Medium – Where good ideas find you. [online]. Dostupné z: <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58> [↑](#footnote-ref-3)
4. Úvod do Blazor pro vývojáře webových formulářů ASP.NET | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 10.04.2021]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/architecture/blazor-for-web-forms-developers/introduction> [↑](#footnote-ref-4)
5. What's the Difference Between Blazor vs Razor. Telerik & Kendo UI - .NET Components Suites & JavaScript UI Libraries [online]. Copyright © 2021, Progress Software Corporation and [cit. 10.04.2021]. Dostupné z: <https://www.telerik.com/blogs/difference-between-blazor-vs-razor> [↑](#footnote-ref-5)
6. What is Blazor? – Blazor University. Blazor University – Learn the new .NET SPA framework from Microsoft [online]. Copyright © 2019 [cit. 10.04.2021]. Dostupné z: <https://blazor-university.com/overview/what-is-blazor/> [↑](#footnote-ref-6)
7. Electron, 1. část - Kutáč.cz . Portfolio Kutáč.cz [online]. Copyright © [cit. 05.04.2021]. Dostupné z: <https://www.kutac.cz/weby-a-vse-okolo/electron-1-cast> [↑](#footnote-ref-7)
8. Build a Desktop App with Electron... But Should You? - YouTube. YouTube [online]. Copyright © 2021 Google LLC [cit. 05.04.2021]. Dostupné z: <https://www.youtube.com/watch?v=3yqDxhR2XxE> [↑](#footnote-ref-8)
9. Databáze. Databáze [online]. Copyright © 2010 Misha [cit. 10.04.2021]. Dostupné z: <http://www.databaze.chytrak.cz/modely.htm> [↑](#footnote-ref-9)
10. OPPEL, Andrew J. Databáze bez předchozích znalostí. Přeložil David KRÁSENSKÝ. Brno: Computer Press, 2006. ISBN 80-251-1199-7 [↑](#footnote-ref-10)
11. OTTE, Lukáš. Databázové systémy [online]. ©2013 [cit. 2021-04-11]. Dostupné z: <http://eknihy.knihovna.cz/kniha/elektronicke-informacni-zdroje> [↑](#footnote-ref-11)
12. Oracle vs MSSQL | 22 Important Comparison You Should Know. Best Online Training & Video Courses | eduCBA [online]. Copyright © 2020 [cit. 11.04.2021]. Dostupné z: <https://www.educba.com/oracle-vs-mssql/> [↑](#footnote-ref-12)
13. HERNANDEZ, Michael J. Návrh databází. Praha: Grada, 2006. Profesionál. ISBN isbn80-247-0900-7. [↑](#footnote-ref-13)