VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTÉM PRO SPRÁVU RODOKMENŮ

GENEALOGY DATABASE SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN CHLÁDEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Chládek Martin**
Program: Informační technologie
Název: **Systém pro správu rodokmenů**
Genealogy Database System
Kategorie: Databáze

Zadání:

1. Seznamte se s problematikou práce s genealogickými daty a s existující databází přepisovaných matričních záznamů ČR. Tato výchozí databáze vzniká v rámci projektu na FIT.
2. Navrhněte a realizujte databázi reflektující potřeby ukládání rodokmenů (vývodů i rozrodů). Databáze musí umět zachytit informace a vztahy, o kterých víme, že jsou platné s určitou mírou pravděpodobnosti v důsledku chyb a neúplných informací ve zdrojových záznamech. Konkrétní osoby z rodokmenu musí být mapovány na záznamy výchozí databáze.
3. Navrhněte a realizujte uživatelské rozhraní pro správu rodokmenů, především vkládání relevantních osob do rodokmenů a jejich vyhledávání.
4. Navržený systém umožní správu rodokmenů jednotlivým uživatelům zvlášť. Rodokmeny se mohou prolínat (např. společný předek) a mohou tak vznikat konflikty, které musí systém odhalit a řešit.
5. Na demonstračních datech otestujte systém a diskutujte jeho další vývoj.

Literatura:

- Moravský Zemský Archiv. Digitalizace archivních fondů. http://www.mza.cz/a8web/a8apps1/a8apps1.htm.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kočí Radek, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Tato bakalářská práce vznikla jako rozšíření projektu DEMoS, který si klade za cíl vytvoření systému pro správu digitalizovaných záznamů starých matrik. Cílem této práce je vytvořit aplikaci, která uživatelům umožňuje spravovat své vlastní rodokmeny. Aplikace automaticky provádí komparace rodokmenů a vyhodnocuje jejich případný nesoulad vedoucí ke vzniku konfliktů. Snahou systému je osoby vložené do rodokmenů provázat se záznamy z databáze digitalizovaných matrik. Výsledný program řeší určitou míru nejistoty ve vztazích mezi osobami a uvažuje proměnlivou kvalitu původních matričních záznamů. Vytvořené řešení poskytuje uživatelům možnosti odhalit nepřesnosti v jejich vlastních rodokmenech, poznat blíže svůj rod a dozvědět se více o svých předcích, díky existenci jejich záznamů v matrikách. Systém je realizován jako webová aplikace a je implementován užitím především následujících technologií: ASP.NET Core, C#, Angular, MySQL, HTML, SCSS.

Abstract

This bachelor thesis originated as an extension to project DEMoS, which aims to create a system for managing digitalized records of old parish books. The goal of this bachelor thesis is to develop an application for managing family trees for its users. The application automatically compares related family trees to find any inconsistencies or conflicts. The system attempts to connect persons in family trees to their corresponding counterparts from the database of registry records. The completed application deals with some amount of uncertainty between persons and takes the various quality of original records into account. The developed solution provides its users the ability to discover inaccuracies in their family trees and learn more about their lineage thanks to the data from registry records. The system is implemented as a web application using mainly the following technologies: ASP.NET Core, C#, Angular, MySQL, HTML, SCSS.

Klíčová slova

genealogie, serielní prameny, C#, .NET Core, ASP.NET Web API, MySQL, Angular, webová aplikace

Keywords

genealogy, seriel sources, C#, .NET Core, ASP.NET Web API, MySQL, Angular, web application

Citace

CHLÁDEK, Martin. *Systém pro správu rodokmenů*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Systém pro správu rodokmenů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Chládek
24. srpna 2020

Poděkování

Tímto bych rád poděkoval mému vedoucímu práce panu Ing. Radkovi Kočímu, Ph.D. za věcné rady, čas strávený konzultacemi a za odborné vedení mé bakalářské práce.

Obsah

1 Úvod **3**

2 Požadavky na systém **5**
2.1 Genealogie 5
2.1.1 Rozrod 6
2.1.2 Vývod z předků 6
2.1.3 Rodokmen 7
2.1.4 Matriky 9
2.2 Funkční požadavky 11
2.3 Existující řešení 12
2.3.1 MyHeritage 13
2.3.2 Ancestry 13
2.3.3 LetMeTree 13
2.3.4 Famberry 14

3 Použité technologie **15**
3.1 Webová aplikace 15
3.1.1 Architektura klient–server 16
3.1.2 Třívrstvá architektura 17
3.2 Front-end 18
3.2.1 HTML 18
3.2.2 SASS 19
3.2.3 TypeScript 20
3.2.4 Angular 21
3.2.5 Knihovna dTree 22
3.3 Back-end 23
3.3.1 Platforma .NET 23
3.3.2 C# 27
3.3.3 MySQL 28
3.4 Další použitý software 28
3.4.1 Git 28
3.4.2 XAMPP 28
3.4.3 Postman 29

4 Návrh systému **30**
4.1 Návrh databáze 30
4.2 Návrh architektury aplikace 34
4.3 Návrh uživatelského rozhraní 35

5 Implementace **40**
5.1 Datová vrstva 40
5.2 Aplikační vrstva 41
5.3 Prezentační vrstva 48
5.4 Testování 53

6 Závěr **54**

Literatura **56**

A Instalace a spuštění **61**

Kapitola 1

Úvod

Tato bakalářská práce byla vytvořena v rámci výzkumného projektu DEMoS[1] zabývajícího se digitalizací a zpracováním údajů ze serielních historických pramenů. Zmíněný projekt je řešen na Fakultě informačních technologií Vysokého učení technického v Brně.

Pro vytvoření rodokmenu je potřeba znát dobře své předky. Většina lidí však nezná mnoho generací svých předků, někteří z nich dokonce ani svého rodiče. Při sestavování rodokmenu je tak potřeba pátrat v matričních záznamech. Vytvořený systém bude umožňovat automatické provázání osob s digitalizovanými matričními záznamy vytvořenými v rámci projektu DEMoS. Matriky tak budou přímo dostupné a uživatel s nově nabytými informacemi může pokračovat v zaznamenávání předků.

Práce si klade za cíl vytvořit webovou aplikaci, která umožní přihlášeným uživatelům vytvářet a spravovat své vlastní rodokmeny. Od systému se očekává, že bude schopen uchovat data genealogických schémat, jež uživatelé v aplikaci budou moci vytvářet. Dále je nutné uvažovat neurčitost ve vztazích mezi osobami a schopnost navázat osoby na matriční záznamy. Za tímto účelem musí být navržena a realizována vyhovující databáze. Při sestavování rodokmenů bude umožněno využívat informace z již existujících rodokmenů, pokud to však uživatelské nastavení povoluje. Ke snížení nejistoty uživatelů a dosažení co nejpravděpodobnějších rodokmenů bude aplikace disponovat jejich porovnáváním a detekcí konfliktů. Aplikace konflikty vyhodnotí a pobídne příslušné uživatele k jejich řešení. Sdílení dat mezi uživateli může současně pomoci při validaci informací, neboť je pracováno mnohdy s neúplnými či chybnými zdroji. Dalším dílčím cílem systému je vyhodnocovat údaje nově přidaných osob a přiřazovat jim potenciální digitalizované matriční záznamy. Tyto serielní prameny poskytuje aplikace DEMoS a přístup k nim bude umožněn skrze grafické uživatelské rozhraní navržené a realizované pro tuto aplikaci.

Obsah dokumentu je strukturován do jednotlivých kapitol. Tento projekt pracuje s termínem rodokmen, ten je však v této práci užíván v jeho laickém významu. Striktní definice pojmu, porovnání a vymezení ostatních genealogických termínů souvisejících s prací je popsáno v kapitole 2. Zmíněná kapitola dále vymezuje podrobnější požadavky na vytvářený systém. V jejím závěru je pojednáváno o již existujících konkurenčních nástrojích pro tvorbu rodokmenů a rozdílech oproti implementované aplikaci. Kapitola 3 řeší teoretické základy webových aplikací a mnou zvolené technologie, kterým dominuje především aplikační rámec ASP.NET Core, Angular a jazyk C#. Rovněž zde lze najít stručné představení programů, které byly během implementace použity. V následující kapitole (kapitola 4) jsou popsány

[1] Podrobnosti ke genealogickému projektu dostupné na: https://www.fit.vut.cz/research/project/1204/ Aplikace DEMoS je dostupná na: http://perun.fit.vutbr.cz/DEMoS/

postupy, které byly užity při návrhu systému. Je zde pojednáváno o návrhu databáze, architektuře vytvořené aplikace a navrženém grafickém uživatelském rozhraní. Způsob strukturování kódu, komunikace mezi jednotlivými vrstvami a jiné důležité dílčí části spojené s implementací jsou popsány v závěrečné kapitole 5.

Kapitola 2

Požadavky na systém

Uvedená kapitola je rozčleněna na tři bloky a řeší zejména implementované principy, nároky kladené na vytvářený systém a odlišnost od konkurenčních systémů.

Záměrem této bakalářské práce je vytvořit a důkladně popsat webovou aplikaci s genealogickou podstatou, kdy nejvíce je v kontextu genealogie pracováno s termínem rodokmen. Jeho význam v této práci, striktní definice a související pojmy důležité k pochopení řešené problematiky v oboru genealogie jsou uvedeny a blíže vysvětleny v první části této kapitoly (2.1). Následující druhá část této kapitoly představuje vymezení funkčních požadavků, které jsou kladeny na implementovanou aplikaci. Tyto požadavky jsou podrobně popsány v sekci 2.2. Poslední sekce 2.3 uvádí již existující dostupná řešení a jejich vzájemnou analogii s vytvořenou aplikací.

2.1 Genealogie

Geneaologie je stará vědní disciplína zkoumající dějiny a vztahy jednotlivých rodů. Je řazena do takzvaných pomocných věd historických. Název pochází z řeckého jazyka a je složeninou dvou slov, vznikl spojením slova „génos“ nebo „genus“, v překladu „rod“, a logos, jehož význam by se dal vyložit jako slovo či pojem. Synonymem pro genealogii je v českém jazyce termín rodopis. [18]

Zejména v historii byla tato vědní disciplína důležitá i pro plnění politických ambicí, kdy příslušnost k danému rodu určovala společenské postavení jedince a oprávnění k jistým činnostem. [38]

Jejím předmětem je tedy studium a mapování lidských rodových linií. Vyzkoumaná data je ale potřeba zaznamenat. Za tímto účelem v genealogii existují schémata určená pro jejich zápis, někdy nazývaná jako genealogické tabulky. Způsoby, jakými popisovat vztahy jedinců v rodu, jsou popsány v následujících podkapitolách.

Na úvod do genealogických schémat je pro správné pochopení potřeba znát termín střen, neboli probant. Střen je osoba, která je při tvorbě schématu tou výchozí, je tedy první zaznamenanou, a od ní se dál vyvíjí dané větvení.

V některých případech se v diagramech používá vizuální rozlišení pohlaví. Například při studiích dědičnosti osob nejsou jména osob relevantní a používá se rozlišení pohlaví pouze tvarem, jenž pohlaví dané osoby reprezentuje. V takových případech poté bývá tvar doplněn informacemi podstatnými pro daný obor. Obecně však platí, že se muž značí čtvercem a žena kruhem, jako je tomu i v uvedených příkladech u jednotlivých schémat [59].

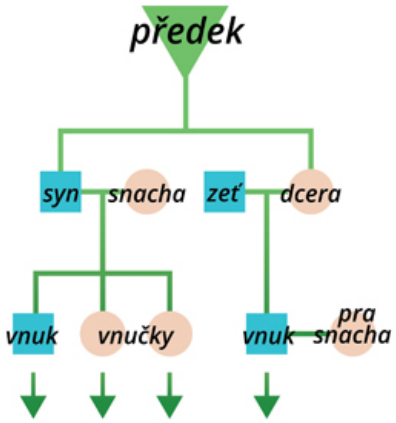
2.1.1 Rozrod

Rozrod je jedním ze způsobů jak zaznamenávat vývoj rodu. Střenem je v tomto případě nejstarší osoba rodu, která je považována za jejího zakladatele [7]. Ve skutečnosti však často bývají výchozí osoby dvě, svázány manželským svazkem, a z jejich pozice jsou zkoumáni všichni jejich přímí potomci. Je tedy vytvářena descendentní linie (od předka k jeho potomkům) [30].

Do rozrodu náleží všichni potomci (muži i ženy) poutáni pokrevenstvím k zakladatelům rodu a vyskytují se tedy linie po meči i po přeslici. Dále do rozrodu spadají osoby pojené manželskými svazky k přímým potomkům a nemanželské děti synů i dcer a jejich potomci. U všech osob v rozrodu je uvedeno křestní jméno i příjmení [38].

Uplatnění rozrodů dopadá dále za hranice popisu historie, v minulosti sloužily například k doložení urozenosti majitele, dnes se využívají i při studiu klinické genetiky, dlouhověkosti a demografie [38].

Základní podstata grafického zápisu rozrodu je vyjádřena na obrázku 2.1.



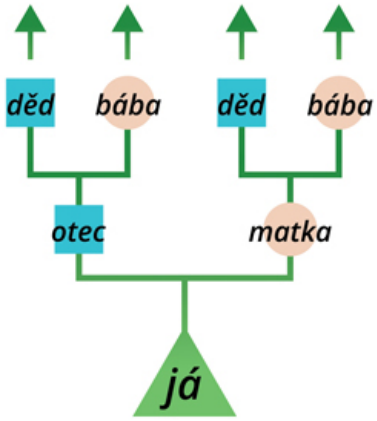
Obrázek 2.1: Schéma pro grafický zápis rozrodu (zdroj: [30])

2.1.2 Vývod z předků

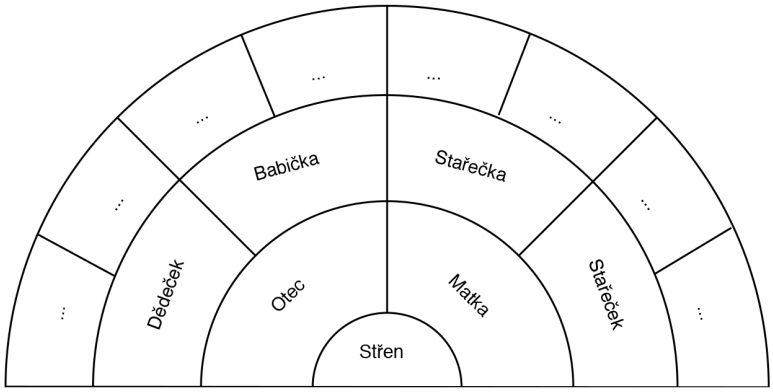
Vývod z předků je schéma značení rodové linie, u kterého je pozornost zaměřena na předky střena. Při snaze vytvořit vývod se postupuje od jedné konkrétní osoby do minulosti, přičemž se zjišťují pouze přímí předci, a to jak po mužské (agnátní) linii, tak i po ženské (kognátní) linii [7]. Snažíme se tedy nalézt rodiče střena, prarodiče, rodiče prarodičů, předky praprarodičů a dále co nejhlouběji do minulosti.

Vzhledem k faktu, že počet předků se s každou generací zdvojnásobí, je možné vypočítat počet osob v určité generaci [7].

Vývod z předků se nejčastěji zaznamenává do stromové struktury (obrázek 2.2) nebo vějířového grafu (obrázek 2.3), jejich grafické zobrazení je uvedeno níže.



Obrázek 2.2: Schéma stromové struktury vývodu z předků (zdroj: [30])



Obrázek 2.3: Schéma vějířového grafu vývodu z předků (zdroj: autor)

2.1.3 Rodokmen

Ačkoliv si pod slovem rodokmen v nejširším slova smyslu velké množství lidí vybaví jakékoliv uspořádání vzájemně příbuzných jedinců, jedná se o konkrétní variantu schématu s pevně stanovenými pravidly.

Rodokmen je základním nástrojem pro vytváření rodové kroniky. Zpracovávána je pouze mužská (agnátní) linie, tedy pouze pokrevenství po meči. Při jeho realizaci je postupováno od zakladatele rodu sestupně po jednotlivých generacích přímých potomků. Zaneseni jsou všichni synové a dcery, jež jsou manželskými potomky mužů patřících do rodokmenu [38].

Zakladatelem rodu (a zároveň střenem) se rozumí nejstarší přímý předek, o kterém je možné dohledat informace. Pouze jeho příjmení je v rodokmenu uvedeno. Za předpokladu, že nějaký z mužských potomků nabyl jiného příjmení, je toto příjmení uvedeno pouze u jeho prvního nositele [38].

Do rodokmenu tedy jsou zaznamenáni [30][38]:

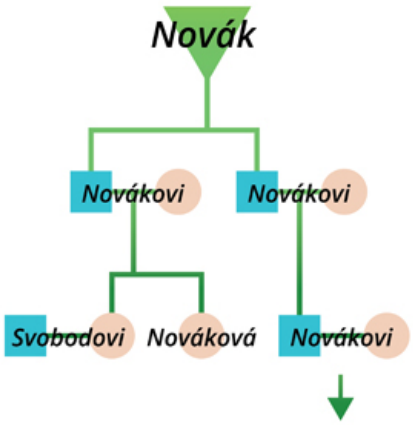
- Všichni muži, jež jsou přímými potomky a to i za předpokladu, že jim byla povolena změna jména či přijali jméno nevěstino
- Dcery mužů, pouze však do první generace (výjimku tvoří nemanželské děti žen, v takovém případě spadají do rodokmenu i ony)
- Osoby v manželském svazku s přímými potomky

Do rodokmenu naopak nepatří [38]:

- Potomci vdaných dcer (ani v případě, že vzniklá rodina přijala sňatkem rodičů příjmení nevěstino)
- Nemanželské děti synů
- Děti osvojené do pěstounství
- Děti přivdaných žen, které byly počaty mimo manželství uvedené v rodokmenu (tedy potomci z jiných manželství či nemanželské děti)

Umísťování potomků do rodokmenů se řídí tradičním patriarchálním modelem rodiny (muž je zakladatelem rodiny) doplněného o institut manželství. V případě, že žena porodí děti, když není součástí manželského svazku, tito potomci stále patří do rodokmenu otce matky. Pokud se však vdá a zplodí potomky se svým manželem, děti už rozšiřují rodokmen rodu otce [30].

Na obrázku 2.4 jsou uvedeny základní principy rodokmenu a jeho zápis v genealogickém schématu.



Obrázek 2.4: Stromový graf rodokmenu (zdroj: [30])

V této bakalářské práci je termín rodokmen používán v přeneseném významu slova a nesplňuje požadavky striktní definice. Přeneseným významem se rozumí výklad, jak pojem rodokmen chápe laická veřejnost.

Tato práce tedy chápe rodokmen jako kombinaci rozrodu a vývodu, kdy střen se stává irelevantním. V takovém případě je tedy střen pouze první zadanou osobou, na kterou

nejsou kladeny žádné nároky a příbuzenské vztahy jsou od něj rozvíjeny libovolně po agnátní i kognátní linii.

V tomto pojetí mohou být zahrnuti všichni přímí potomci a předkové nezávisle na pohlaví. Za zakladatele rodu může být považována i žena a větve rodokmenu mohou tvořit rodové linie po meči i po přeslici. Ve vyvinuté aplikaci jsou omezení při tvorbě stromu života minimální a uživateli je při jeho realizaci poskytnuta volnost. Více k možnostem v aplikaci a omezeních akcí uživatele je řešeno v kapitole 2.

2.1.4 Matriky

Matrikami se v České republice rozumí veřejné úřední knihy, které evidují základní životopisné údaje o občanech. Tyto knihy se nazývají „matriční knihy“ a jsou uloženy na matričních úřadech [31].

Jsou zaznamenávány tři druhy událostí, legislativa České republiky definuje matriky v § 1 zákona č. 301/2000 Sb. [54] následovně: „*Matrika je státní evidence narození, uzavření manželství a úmrtí fyzických osob na území České republiky a narození, uzavření manželství a úmrtí, k nimž došlo v cizině, jde-li o státní občany České republiky.*“

Záznamy do matričních knih provádějí způsobilí a pověření zaměstnanci matričního úřadu. Každý ucelený zápis události do sbírky listin je matričním záznamem. Zápisy do matričních knih jsou dodnes chronologicky zapisovány rukopisně do předem svázaných knih [54].

Příslušný typ záznamu je evidován v přiřazené matrice, existují [54]:

- *Kniha narození* – zapisuje se
  (a) jméno, příjmení, rodné číslo a pohlaví dítěte
  (b) datum, místo narození dítěte
  (c) jména, příjmení, data a místa narození, rodná čísla, státní občanství a místa trvalého bydliště obou rodičů
- *Kniha manželství* – zapisuje se
  (a) datum a místo uzavření manželství
  (b) dohoda novomanželů o příjmení
  (c) osobní údaje (jméno, příjmení, rodné číslo) obou manželů, navíc jejich data a místa narození, osobní stavy a státní občanství
  (d) jména, příjmení, data a místa narození rodičů manželů
  (e) osobní údaje svědků
- *Kniha úmrtí* – zapisuje se
  (a) datum a místo úmrtí i narození
  (b) osobní údaje, pohlaví, osobní stav, státní občanství a místo trvalého pobytu zemřelého
  (c) případně osobní údaje pozůstalého manžela/manželky

Vyhláška poslanecké sněmovny ze dne 13. června 2006, Předpis 300/2006 Sb. [55], § 2 popisuje zavedení čtvrté knihy:

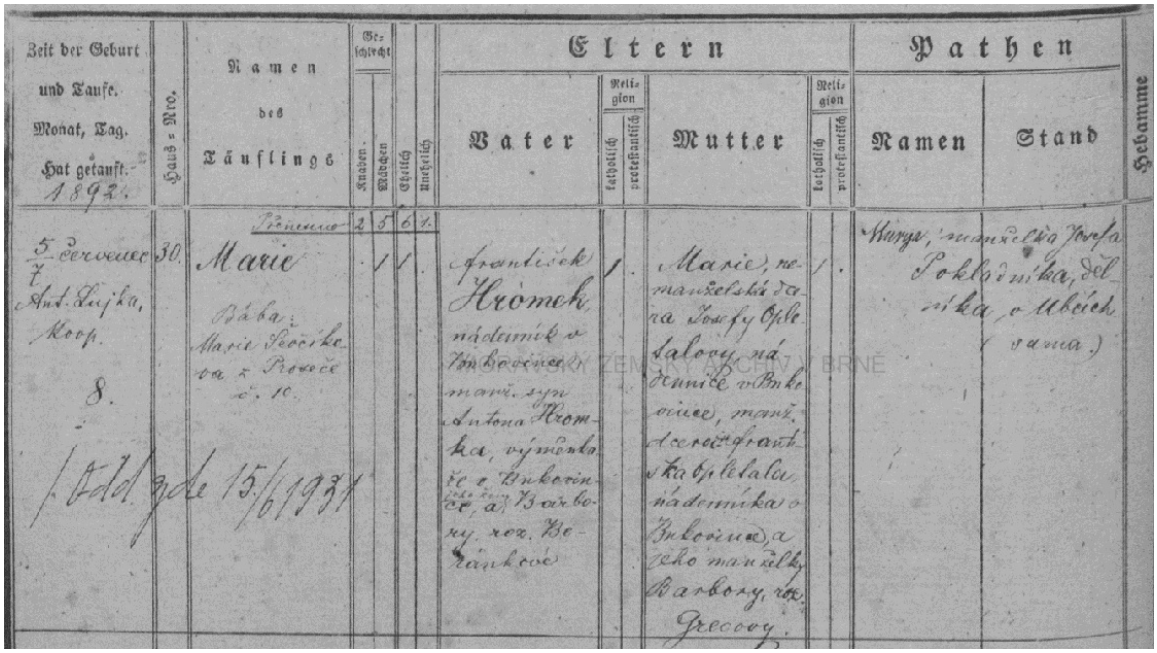
- *Kniha registrovaného partnerství* – zapisuje se
  (a) jméno a příjmení, datum a místo narození, rodné číslo, osobní stav a státní občanství obou partnerů stejného pohlaví
  (b) jména, příjmení, data a místa narození rodičů partnerů

Ke každému matričnímu záznamu je na závěr připsáno datum zápisu a podpis matrikáře (odpovědného úředníka).

Matriční knihy jsou typem serielních historických pramenů a základním pramenem informací při genealogickém výzkumu. Zápis záznamu je doprovázen vydáním příslušného matričního dokladu (rodný, oddací a úmrtní list a doklad o registrovaném partnerství), který obsahuje shodné údaje se zápisem v matriční knize [56].

V této práci je však vycházeno z matričních záznamů, které vznikly převážně v devatenáctém století na území Čech, Moravy, Slezska a pohraničí. Struktura matričních záznamů před rokem 1900 byla odlišná od té dnešní. Například informace o rodném čísle jedinců úplně chybí, neboť tento jednoznačný identifikátor byl zaveden na území českého státu později.

Ukázka matričního záznamu z tohoto období je k vidění na obrázku 2.5.



Obrázek 2.5: Matriční záznam z 19. století (zdroj: [1])

2.2 Funkční požadavky

Cílem je realizovat aplikaci, která umožňuje uživateli vytvářet si vlastní rodokmeny. Uživatelské rodokmeny jsou vzájemně mezi sebou kontrolovány a generují interakci, která vyžaduje obsluhu uživatele. Podrobněji jsou tyto nároky na aplikaci popsány níže.

Význam termínu rodokmen v této práci je blíže vysvětlen v předchozí kapitole (podkapitola 2.1.3), včetně rozdílů oproti striktní definici. Při vytváření rodokmenů je v aplikaci uživatel omezen minimálně, vytvářet však lze pouze rodokmen pro jeden rod současně. Systém se snaží zabraňovat nereálným skutečnostem kontrolou uživatelských vstupů (sňatek stejných pohlaví a podobně).

Vzhledem k faktu, že žádné ze zmíněných genealogických schémat při sestavování nebere v potaz registrovaná partnerství, i v pojetí této práce je možné mezi osobami v rodokmenu vytvořit pouze manželský vztah, nikoliv registrovaná partnerství.

V implementovaném systému má každý uživatel možnost vytvořit více rodokmenů, jelikož může znát více odvětví své rodiny, které ale nedokáže vzájemně propojit. Dalším uvažovaným scénářem je sestavování rodokmenu pro osobu blízkou nebo sestavování rodokmenu na zakázku pro cizí osobu.

Při zadávání nové osoby do rodokmenu je uživatel vyzván k vyplnění základních osobních údajů přidávané osoby. Na základě těchto údajů je prohledávána databáze aplikace se snahou najít záznamy osob s podobnými údaji. Mohou nastat dvě možnosti výsledků vyhledávání:

1. *Osoba už v aplikaci existuje* – v databázi aplikace již existuje osoba totožná s vyhledávanou (byla tedy v minulosti vytvořena jiným uživatelem). V takovém případě, pokud ji uživatel zvolí, není vytvářen nový záznam a dotyčná osoba je pouze přidána do příslušného rodokmenu.

2. *Osoba vyhovující požadavkům neexistuje* – uživatel má jedinou možnost, kterou je nechat vytvořit nový záznam v databázi aplikaci.

Autor tohoto požadavku je pobízen k vybrání některého z výsledku vyhledávání. Pokud je přesvědčen, že žádná z nabízených osob není totožná s jeho navrhovanou (nebo nebyly nalezeny žádné výsledky), může zvolit variantu popsanou v posledním bodě.

Snahou aplikace je uživatelem zadané osoby propojit s existujícími matričními záznamy z databáze digitalizovaných matrik. Toto propojení je možné na základě shody zadaných údajů s údaji osoby v digitalizovaných matrikách. Aplikace následně uchovává informaci umožňující referenci na původní matriční záznam. Pokud takový záznam existuje, uživatelské rozhraní nabídne odkázání uživatele na webovou stránku s kompletním záznamem, tak jak byl zapsán v matriční knize. Pokud není matriční záznam dostupný, bude uživatel odkázán na seznam potenciálních osob.

Zmíněná webová stránka s kompletními informacemi a databáze digitalizovaných matrik ale nejsou předmětem této práce, implementovaná aplikace se na ně pouze odkazuje. Tyto komponenty vznikají jako projekt souběžně s prací na Fakultě informatiky Vysokého učení technického v Brně[1].

Mezi rodokmeny jednotlivých uživatelů probíhá vzájemné porovnávání pro odhalení případného nesouladu. Tento nesoulad je nazýván jako kolize a vzniká, když pro stejný vztah, který může existovat jen jednou, existuje záznam pro dvě a více osob. Kolize mohou vznikat dvojího typu:

[1] Více informací ke genealogickém projektu na FIT VUT: https://www.fit.vut.cz/research/project/1204/

- *různý předek* – nastává ve chvíli, kdy pro stejnou osobu existuje napříč rodokmeny jiný předek. Jinak řečeno, když se autoři rodokmenů neshodnou na matce nebo otci dotyčné osoby.

- *kolize generací* – vzniká, když autor některého z rodokmenů definuje mezi párem osob manželský vztah, zatím co v jiném rodokmenu je mezi těmito osobami definován vztah předek–potomek.

V případě vzniku konfliktu je každý uživatel, jenž danou osobu má ve svém rodokmenu, na tuto skutečnost upozorněn a je vyzván k jeho řešení. Každý uživatel může mít pro kolizi jiné řešení a databáze aplikace to musí respektovat.

Porovnávání nemusí nutně probíhat mezi všemi rodokmeny. Při zakládání nového rodokmenu uživatel vybírá ze tří různých typů izolace svého rodokmenu. Na základě zvoleného typu je poté povoleno porovnávání a nahlížení ostatních uživatelů. Tyto typy jsou:

- *veřejný* – zadané osoby v rodokmenu jsou porovnávány s ostatními rodokmeny a funguje zaznamenávání kolizí, do veřejného rodokmenu má možnost nahlédnout cizí uživatel a případně si jej zkopírovat do svého.

- *neveřejný* – odbourává poslední zmíněnou funkcionalitu, do rodokmenu může tedy nahlédnout pouze jeho vlastník, zároveň však stále probíhá porovnávání pro případné odhalení kolizí.

- *privátní* – úplně eliminuje porovnávání a nahlížení cizích uživatelů

Uživatel si o každé osobě ve svém rodokmenu může zobrazit podrobnější informace, včetně seznamu všech veřejných rodokmenů, ve kterých se osoba vyskytuje. Do těchto rodokmenů je mu nabídnut náhled, nemá však možnost s nimi nijak manipulovat. Pokud by chtěl některý z veřejných rodokmenů ostatních uživatelů upravovat, může si jej zduplikovat a propojit s vlastním rodokmenem.

V aplikaci je pracováno s informacemi, které jsou známy s určitou mírou nejistoty. Zejména se jedná o reference na matriční záznamy. Pro údaje jedné osoby může existovat v databázi matrik více různých záznamů, přičemž nelze rozhodnout, která osoba je ta správná. Limit pro aplikaci je omezen na maximálně pět potenciálních osob. Pokud je osob více, uživatelské rozhraní nenabídne přesměrování na konkrétní matriční záznamy, ale pouze na seznam osob se shodnými údaji.

2.3 Existující řešení

V současné chvíli existují vyhledávače v digitalizovaných matrikách a rovněž i systémy pro tvorbu a správu vlastních rodokmenů. Čím se však mé řešení od většiny nástrojů liší je fakt, že tyto dva přístupy kombinuje. Pro každou nově přidanou osobu do rodokmenu jsou automaticky vyhledávány matriční záznamy, což značně zjednodušuje postup vytváření rodokmenu. Běžně by uživatel musel nejprve vyhledávat matriční záznamy a v zápětí zjištěné údaje přepisovat do aplikace.

Vyvinutá aplikace je navíc orientována a cílena na Českou republiku, neboť pracuje s matričními záznamy českých občanů.

Zároveň disponuje jednoduchým, přehledným a intuitivním ovládáním.

2.3.1 MyHeritage

Prvním konkurentem je rozsáhlá komplexní platforma s názvem *MyHeritage*[2], která nabízí uživateli možnost vytváření vlastních rodokmenů, koupi a vyhodnocení DNA testů, a další.

Tento projekt nabízí tři různé verze předplatného, prvním je verze „basic“, což je základní balíček (zdarma), který umožňuje bezplatné vedení účtu, vytváření rodokmenu až do 250 osob nebo 500MB dat. Do rodokmenu umožňuje i nahrát DNA testy jednotlivých osob, jejich fotky, podrobné informace, životopisy a podobně. Placenými verzemi jsou poté předplatné „Premium“ a „Premium Pro“, které vylepšují a přidávají některé možnosti.

Nástroj stejně jako implementovaná aplikace nabízí analýzu rodokmenů a hledání shodných osob napříč rodokmeny nebo spojování osob s historickými dokumenty.

Izraelská společnost *MyHeritage ltd.* rovněž nabízí i nativní aplikaci pro tvorbu rodokmenů. Ta se od té webové funkcionalitou neliší, pouze v případě pozvání jiných uživatelů k úpravám je využita webová verze, aby byla odstraněna nutnost instalovat desktopový program na počítači pozvaných osob.

Celkově se jedná o velmi rozsáhlý komplikovaný profesionální nástroj pro realizaci genealogických schémat, kde je možné uložit velké množství informací.

2.3.2 Ancestry

Druhým známým nástrojem pro tvorbu rodokmenů je aplikace zvaná *Ancestry*[3]. Tato platforma disponuje více než 20 miliardami záznamů. Snaží se identifikovat zadané osoby a dosadit je do kontextu svých záznamů. Na základě těchto informací poté vyhodnocuje pravděpodobné předky a nabízí jejich data.

Tato platforma ale neposkytuje členství zdarma a ke správě svého rodokmenu je nutné platit měsíční tarif.

2.3.3 LetMeTree

Dalším systémem pro správu rodokmenů je webová aplikace *LetMeTree*[4]. Tento systém je zdarma pro soukromé použití, není ale zdaleka tak rozsáhlý. Umožňuje pouze základní úkony, jakými jsou přidání a odebrání vrstevníka, potomka či předka. Při přidávání osoby probíhá vyhledávání s cílem nalézt již existující záznam, stejně jako u předchozích konkurentů.

Nevýhodou je chybějící podpora zobrazení rodokmenu v grafické podobě stromu, data jsou promítána do tabulky nebo seznamu. Uživatelské rozhraní je navíc zastaralé a nechlubí se ani přehledností. Přihlášený uživatel je omezen na správu pouze jediného rodokmenu, který však je schopen vypisovat názvosloví rodinných příslušníků, kdy jako výchozí osoba může být nastaven kterýkoliv člen rodiny.

[2]Dostupné z: http://myheritage.com
[3]Dostupné z: https://www.ancestry.com
[4]Dostupné z: http://www.letmetree.com

2.3.4 Famberry

Poslední zmíněný konkurent nese název *Famberry*[5]

V porovnání s předchozími systémy se tento liší tím, že vyžaduje před samotným vytvářením rodokmenu volbu uživatele, zda chce definovat rod po agnátní (mužské) linii nebo po ženské (kognátní) linii. Tyto přístupy nelze později kombinovat.

Aplikace je určena především pro skupinovou kooperaci více členů rodiny při sestavování rodokmenu. Důkazem je i přítomný seznam změn, které konkrétní uživatel provedl, a integrovaný chat, pomocí kterého mohou spoluautoři mezi sebou komunikovat.

Každý uživatel může zdarma vytvořit/spolupracovat pouze na jednom rodokmenu, na vývoji více rodokmenů je možné se podílet pouze s předplatným tarifem „Premium“.

[5]Dostupné z: https://www.famberry.com

Kapitola 3

Použité technologie

Tato kapitola si klade za cíl představit vybrané technologie použité při implementaci.

Na úvod této části práce jsou vysvětleny obecné principy tvorby webových aplikací, jež jsou zároveň použity při implementaci této práce (3.1). Obsahem následujících dvou podkapitol je seznámení s programovacími jazyky a technologiemi vybranými k vývoji systému. V závěru této kapitoly je uveden a blíže popsán software, který byl v průběhu vývoje použit.

3.1 Webová aplikace

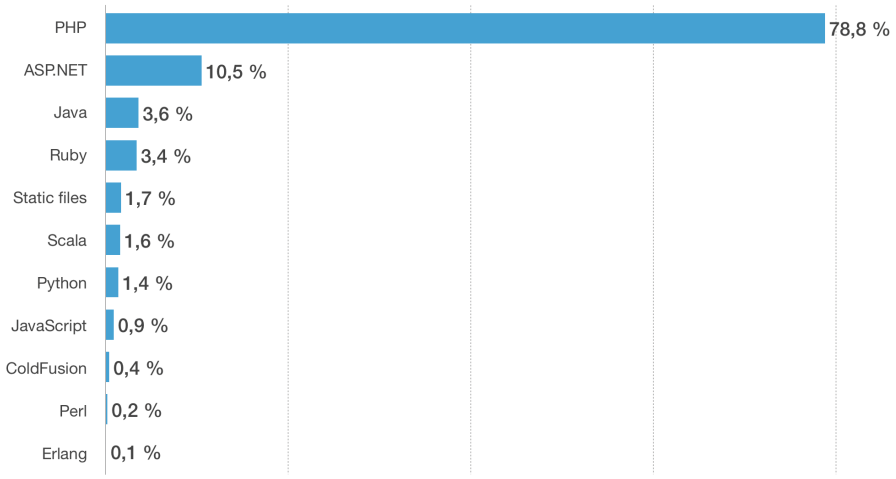
S celosvětovým rozšířením internetu vzniklo velké množství statických webových stránek. Pojem „statické“ v označení znamená, že na klientův dotaz vrátí server webovou stránku přesně tak, jak ji má uloženou. V prohlížeči je tedy její obsah stejný pro každého uživatele, který na webovou stránku přistoupí [57].

S postupem času se stal osobní počítač dostupnějším a mohla si jeho koupi dovolit již i většina domácností. Současně začaly vznikat požadavky na dynamičnost webových stránek a rozšířily se webové aplikace [57].

Webové aplikace na rozdíl od statických webových stránek různě přizpůsobují zobrazovaný obsah konkrétnímu uživateli [57]. Samotnou webovou aplikací se rozumí software, který je spuštěn na straně webového serveru a zajišťuje dynamické vytváření stránky s personalizovaným obsahem na základě HTTP požadavků přijatých od uživatele [43].

K webovým aplikacím je možné přistupovat pouze skrze webový prohlížeč. Základem pro takovouto aplikaci je architektura klient–server, která je podrobněji popsána v podkapitole 3.1.1.

K implementaci takových aplikací, které jsou spuštěny na straně webového serveru, lze použít velké množství odlišných programovacích jazyků, jejich procentuální využití je na k vidění na obrázku 3.1.



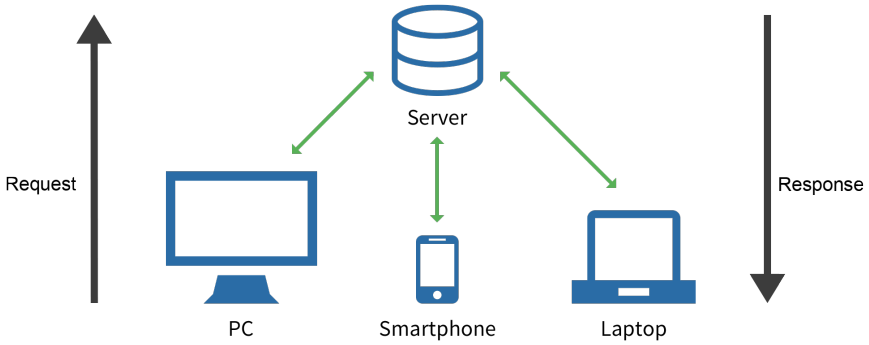
Obrázek 3.1: Technologie používané pro implementaci webové aplikace (zdroj: [50])

K datu psaní práce dominuje mezi programovacími jazyky pro tvorbu webové aplikace jazyk PHP se 78,8 % podílu. Druhého místa se ujímá technologie ASP.NET s 10,5 %.

Jazyk PHP využívají například populární sociální síť *Facebook*, vyhledáváč *Google* či *Yahoo*. Framework ASP.NET implementují weby jako jsou *MSN.com* nebo vyhledávač *Bing*. Třetí nepoužívanější jazyk Java používají pro svůj back-end internetový obchod *Amazon*, *eBay* a služba *YouTube* [19].

3.1.1 Architektura klient–server

Model klient–server je v současnosti nejvíce užívanou síťovou architekturou při implementaci distribuovaných aplikací. Popisuje rozdělení úloh mezi roli klienta a serveru, jež spolu komunikují přes počítačovou síť. Vychází z myšlenky, že data by měla být zpracovávána v místě uložení. Pokud jsou tedy zdroje aplikace uloženy centrálně na serveru, jejich zpracování by měl zajistit rovněž server [32].



Obrázek 3.2: Architektura klient-server (zdroj: [44], upraveno)

V architektuře jsou definovány zmíněné dvě role:

- *Klient* – zasílá požadavky na server, přijímá odpovědi nebo využívá konkrétní služby serveru. Může být připojen k více serverům současně [3]. Klientem je zpravidla uživatelský počítač.

- *Server* – umožňuje přistupovat klientovi ke zdrojům, zpracovává jeho požadavky, zasílá odpovědi a zajišťuje data [3].

Klienti se mohou dále terminologicky dělit. V závislosti na množství obsažené logiky a funkcionality rozlišujeme dva základní typy klientů: tenký a tlustý klient.

Tenký klient se stará pouze o distribuci zobrazení uživateli a neobsahuje žádnou aplikační logiku (vyjma základní validace dat ve webových formulářích). Tenkým klientem je zpravidla internetový prohlížeč. Vzhledem k tomuto faktu existují minimální požadavky na instalaci, konfiguraci a také na výkon zařízení. Další výhodou je aktualizace verzí aplikace, po nasazení aktuálnější verze na server je automaticky ihned přístupná všem uživatelům bez jakéhokoliv jejich zásahu [53].

Tlustý klient je na hardware počítače náročnější, důvodem je, že kromě prezentační vrstvy obsahuje i veškerou aplikační logiku. Připojen je obvykle k databázovému serveru, ze kterého stahuje velký objem dat, s těmito daty pracuje, a poté je přenese zpět na server. Rozdílem je mimo jiné i nezbytná instalace samotného klienta, tím pádem i závislost na operačním systému. Liší se i způsob aktualizací, kdy aktualizaci musí provést každý uživatel na svém lokálním zařízení [53].

3.1.2 Třívrstvá architektura

Model třívrstvé architektury je v aplikacích často používán jako specifická implementace architektury klient–server. Na rozdíl od dvou disjunktních subjektů, které popisuje klient–server, třívrstvá architektura, jak už název napovídá, rozděluje aplikaci do tří definovaných modulů (vrstev). To umožňuje flexibilnější vývoj a údržbu aplikace, kdy upravovat lze pouze jeden modul nezávisle na ostatních [16].

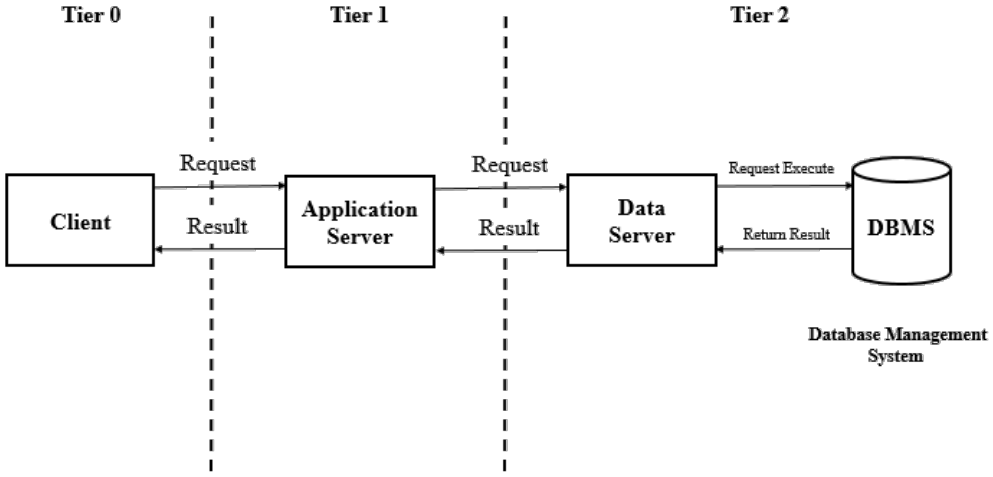
Z hlediska softwaru jsou popisovány následující vrstvy, kdy každá plní svoji specifickou úlohu [11][16][26]:

- *Prezentační vrstva* – zobrazení informací, které je prezentováno uživateli, nejčastěji ve formě grafického uživatelského rozhraní. Může kontrolovat validitu uživatelských vstupů, neobsahuje však žádnou aplikační logiku. Příkladem technologií pro realizaci prezentační vrstvy mohou být jazyky HTML, CSS a JavaScript.

- *Aplikační vrstva* – část implementující veškerou vlastní logiku aplikace, jsou zde konány všechny výpočty, provádí dotazování nad databází. Mezi populární technologie pro programování aplikační vrstvy patří PHP, C#, Java, a Python.

- *Datová vrstva* – reprezentována databází, zajišťuje podporu transakcí a perzistentní ukládání dat. Pro uložení dat lze použít systémy MySQL, MongoDB, SQL server a další.

Rozdělení jednotlivý vrstev a jejich vzájemná komunikace je znázorněna na obrázku 3.3.



Obrázek 3.3: Schéma třívrstvé architektury (tenký klient) (zdroj: [5])

První vrstvou je klientský počítač, přičemž zobrazení webové stránky, které uživatel na svém počítači vidí, zařizuje prezentační vrstva. Komunikace s aplikační vrstvou (spuštěnou na aplikačním serveru) probíhá zpravidla užitím protokolu HTTP/HTTPS. Server poté vrací náležitá data a prezentační vrstva vykonstruuje HTML dokument. Výměna informací mezi aplikačním serverem a databází probíhá iniciováním komunikace ze strany aplikační vrstvy. Ta vytváří a zasílá SQL dotazy, které databázový systém přijme, zpracuje a vrátí odpověď v podobě příslušných dat [26].

3.2 Front-end

Front-end (či „client-side“) je v softwarovém inženýrství pojem, který značí vrstvu aplikace přímo přístupnou uživateli. Jedná se o vše, co člověk vidí při práci s aplikací, včetně jemu zobrazených dat, rozmístění elementů, barev, animací a podobně. Snahou je vytvořit přehledný a líbivý vzhled, který však musí zůstat intuitivní a jednoduchý na použití [35]. Technologie, které byly za tímto účelem použity jsou popsány níže.

3.2.1 HTML

Jazyk HTML („Hyper Text Markup Language“) je značkovací jazyk, který popisuje způsob rozložení jednotlivých elementů na webové stránce. Vychází ze standardizovaného univerzálního značkovacího jazyka SGML („Standard Generalized Markup Language“) [24].

Jazyk je postaven na dílčích elementech, element je poté úsek dokumentu vymezený značkami (anglicky „tags“). Každá značka má jméno a může mít definované atributy a obsah. Obsahem se rozumí další vnořené značky nebo text.

Elementy mohou být dvojího typu [24]:

- *blokové* – začínají na novém řádku a zabírají vymezený prostor na stránce v celé své šířce a délce. Příkladem může být element `<div>`, paragraf `<p>` nebo nadpis 1. úrovně `<h1>`.

- *řádkové* – nezačínají na novém řádku, vymezují si pouze minimální nezbytný prostor. Obvykle se používají k formátování obsahu blokových elementů. Mezi řádkové elementy se řadí ku příkladu element pro odkazy `<a>` či vytváření indexů pomocí `<sup>` a `<sub>`.

HTML je v současnosti dominantní technologií pro vytváření webových stránek a je podporován všemi nejpoužívanějšími prohlížeči napříč počítačovými i mobilními platformami [24].

Obrázek 3.4: Zastoupení značkovacích jazyků u webových stránek[1](zdroj: [49])

Jak ukazuje výše uvedený graf 3.4, HTML je v době psaní práce s 89,5 % nejvyužívanějším značkovacím jazykem při implementaci webových stránek.

V současnosti je poslední majoritní verze označována jako HTML5. Ta byla v říjnu roku 2014 stanovena konsorciem W3C jako doporučená technologie k vytváření konvenčního webu. [17] Tato verze s sebou přinesla velké množství novinek, příkladem může být přímá podpora audia a videa, podpora kaskádových stylů CSS3, vykreslování vektorových obrázků či atribut canvas, umožňující generování grafiky pomocí JavaScriptu [41].

3.2.2 SASS

SASS („Syntactically Awesome Style Sheets“) je kompilovaný jazyk a plní funkci CSS preprocesoru. V době psaní práce webové prohlížeče nepodporují přímo SASS a tak musí být tato syntaxe před spuštěním přeložena do CSS.

CSS je elementární jazyk pro popis zobrazení elementů na webové stránce. Právě jeho možnosti jazyky SASS rozšiřují. Hlavními změnami je možnost implementace vnořování stylů, zavedení proměnných, mixinů, dědičnosti a další. Mimo jiné je kód přehlednější, méně časově náročný a snadněji udržitelný.

Jazyk SASS je dostupný ve dvou různých syntaxích, SASS a SCSS. Protože se jedná o stejný jazyk a liší se pouze syntaxe, funkcionalita zůstává shodná. Syntaxe SASS je o něco přehlednější, nepoužívá složené závorky, ani středníky, ale sází na odsazení obdobně jako programovací jazyk Python. Syntaxe SCSS je naopak velice podobná jazyku CSS, dá se říci, že je rozšířenou syntaxí CSS. V tom spočívá výhoda zejména z hlediska kompatibility, protože každý CSS soubor je validní i v SCSS [6].

[1] Webová stránka může používat kombinaci více značkovacích jazyků.

```
$my-color: #6F9B12

nav
  border: 1px
  padding: 10px
  div
    color: $my-color
```

Výpis 3.1: Syntaxe SASS

```
$my-color: #6F9B12;

nav {
  border: 1px;
  padding: 10px;
  div {
      color: $my-color;
  }
}
```

Výpis 3.2: Syntaxe SCSS

```
nav {
  border: 1px;
  padding: 10px;
}

nav div {
  color: #6F9B12;
}
```

Výpis 3.3: Výsledné CSS

Z kódu je patrné, že syntaxe SCSS je mnohem více podobná konvenčnímu CSS. V tomto příkladě se také vyskytuje ukázka proměnných a vnořování.

3.2.3 TypeScript

TypeScript je open-source programovací jazyk. Byl vydán v roce 2012 firmou Microsoft a je aktivně vyvíjen do dnešních dní. Jazyk jako takový lze chápat jako nadstavbu jazyka JavaScript, každý kód napsaný v JavaScriptu je tedy validní i v TypeScriptu [21].

Rozšiřuje JavaScript o statické typování, rozhraní, dědičnost a další principy z objektově orientovaného programování. Využívá přitom standardu ECMA-262[2] společnosti Ecma International, která definuje specifikace jednotlivých verzí skriptovacího jazyka ECMAScript. Právě z něj TypeScript vychází [46].

Obrázek 3.5: Standardizace jazyka TypeScript (zdroj: [46])

Základy programovacího jazyka TypeScript vychází ze specifikace ECMAScript 5 (ES5), což je zároveň i specifikace, kterou oficiálně využívá JavaScript. Z šesté generace jazyka ECMAScript (ES6) využívá například prvky, jakými jsou třídně orientované programování nebo implementace modulů. Nad rámec navíc jazyk TypeScript implementuje například generické typy a atributy [46].

Kód napsaný v TypeScriptu je překládán do JavaScriptu. K tomu je využíván tak zvaný transpiler, což je druh kompilátoru, který čte kód napsaný v jednom jazyce a produkuje kód v jazyce jiném. Oba kódy mají přitom shodnou funkcionalitu. Transpiler poté převede kód do ES5 nebo v případě potřeby až do ES3 [21]. Během kompilace kontroluje syntaktickou správnost programu a jestliže je nekorektní, vyústí v neúspěšný překlad. Díky tomu redukuje chyby ještě před spuštěním kódu v porovnání s JavaScriptem [46].

[2]Specifikace standardu ECMA-262 dostupná na: https://www.ecma-international.org/publications/standards/Ecma-262.htm

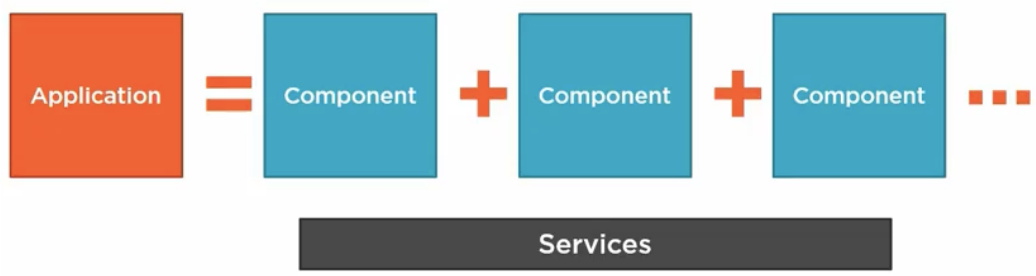
3.2.4 Angular

Angular je framework pro tvorbu front-end vrstvy u moderních webových aplikací. Tato open-source platforma se zaměřuje na tzv. „single-page“ aplikace (zkráceně „SPA“), což jsou dynamické webové stránky, při jejichž používání má uživatel podobnou zkušenost jako by pracoval s nativní aplikací. Celou aplikaci představuje jediná stránka, která je načtena ze serveru pouze jednou a její obsah se dynamicky mění na základě interakce s uživatelem [14].

Vydán byl v roce 2016 a jedná se již o druhou generaci (nikoliv verzi) této platformy. Za jejím vydáním stojí tým vývojářů společnosti Google, který i nadále pracuje na jeho podpoře.

Angular využívá programovací jazyk TypeScript a jazyk HTML. Pro stylování vzhledu aplikace je možné vybírat z více technologií, všechny jsou však založeny na elementárním konceptu CSS stylů [4].

Základy aplikace vytvořené v Angularu tvoří komponentová architektura se službami.

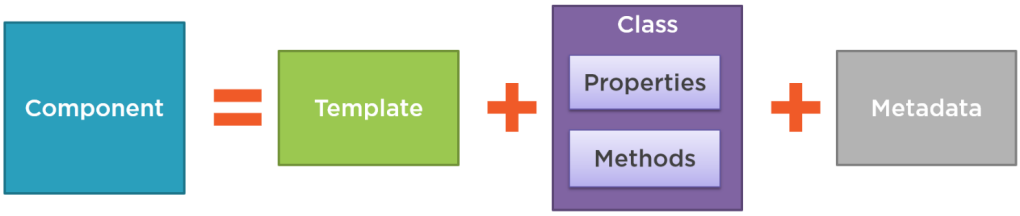


Obrázek 3.6: Stavba aplikace v Angular (zdroj: [23])

Jak je vidět na obrázku 3.6, Angularová aplikace je z velké části tvořena souborem komponent. Základem těchto komponent je třída (viz obrázek 3.7), z toho je patrné, že Angular využívá základních principů objektově orientovaného programování (OOP).

Komponenty nemusí existovat v jedné rovině, ale mohou být vnořovány do sebe a vytvářet stromovou hierarchii. Ve skutečnosti vždy existuje jedna kořenová komponenta, která vytváří hierarchii výsledného HTML souboru a skládá tak objektový model dokumentu (anglicky „Document Object Model“, akronym „DOM“) [4].

Služby (anglicky „services“) poté představují aplikační logiku webu. Jejich nejčastější úlohou je komunikace a distribuce dat mezi klientskou aplikací a serverovým API. Komponenty mohou využívat služby užitím tzv. „Dependency Injection“ [29].



Obrázek 3.7: Struktura komponenty (zdroj: [23])

Každá komponenta sestává ze 3 částí:

- *Šablona* (Template) – šablony kombinují HTML kód se značkami Angularu a umožňují modifikovat HTML elementy ještě před tím, než budou zobrazeny uživateli [4].
- *Třída* (Class) – třída je základem komponenty, každá komponenta implementuje třídu, která může obsahovat data a metody. Pomocí tzv. direktiv je k ní navázána šablona a příslušný styl [29].
- *Metadata* – definovány dekorátorem, který disponuje informacemi o navázané třídě. Na základě doložených informací se pak Angular rozhoduje, zda se jedná pouze o běžnou třídu nebo o komponentu. Dekorátor je v programu identifikován prefixem `@` [4].

Logickým seskupením komponent vznikají tzv. moduly. Angularový modul naváže komponenty s příslušnými službami a vytváří funkční jednotky. Pro tuto konfiguraci následně zajistí kompilační kontext. Každá aplikace implementovaná v tomto frameworku obsahuje alespoň jeden funkční modul, typická aplikace jich však obsahuje několik [4].

Angular vs. AngularJS

Jistou míru nejistoty ve zvolené technologii by mohl způsobit název použitého frameworku, který je velmi podobný platformě AngularJS. Jedná se však o dva odlišné frameworky.

Společným znakem je, že obě technologie jsou webovými aplikačními rámci, zaměřujícími se na tvorbu „single-page“ aplikací. Za vývojem obou platforem stojí společnost Google. Angular je však novější a jedná se o nástupce staršího AngularJS. Mezi sebou jsou vzájemně nekompatibilní.

Moderní Angular staví na jiných principech a na rozdíl od svého předchůdce, který používá JavaScript, pracuje s programovacím jazykem TypeScript. V minulosti byl nazýván jako Angular 2, ale vzhledem k sémantickému značení nových verzí bylo od tohoto označení upuštěno a nyní je znám jako pouze Angular [29].

3.2.5 Knihovna dTree

Jedním ze způsobů vykreslení rodokmenu na webové stránce je použití externí knihovny. dTree[3] je JavaScriptová knihovna, která očekává na vstupu objekt v definovaném formátu a jejím výstupem je graficky zobrazený rodokmen. Pro realizaci vizualizace využívá knihovnu D3[4] („Data-Driven Documents“), která za tímto účelem kombinuje technologie SVG, Canvas a HTML.

Nástroj podléhá licenci MIT, jedná se tedy o open-source projekt, který je možné volně šířit a upravovat, podmínkou je zachování kopie licence a jméno autora.

[3] Dostupné na: https://github.com/ErikGartner/dTree
[4] Dostupné na: https://github.com/d3/d3

3.3 Back-end

Back-end (či „server-side“) je pojem, jenž se v souvislosti s webovými aplikacemi odkazuje na část programového kódu, který umožňuje uživateli s aplikací pracovat, ale nemá k ní přístup. Tato vrstva je odpovědná za práci s daty a obsahuje všechny funkce, které vytvářejí aplikační logiku [35]. Back-end komunikuje s vrstvou front-end (popsána v kapitole 3.2), posílá a přijímá data potřebná ke korektnímu zobrazení informací na webové stránce [13].

K realizaci této vrstvy aplikace lze použít velké množství programovacích jazyků a různých postupů, technologie související s vytvořením této bakalářské práce jsou popsány níže.

3.3.1 Platforma .NET

Platforma Microsoft .NET (též nazývaná pouze jako .NET) je softwarová platforma poskytující prostředky pro vývoj různorodých aplikací. Těmi mohou být aplikace pro Windows, cloudové služby, webové aplikace, hry, aplikace pro mobilní platformy a další [15].

Na nejnižší úrovni architektury se nachází společné běhové prostředí nazvané modul CLR („Common Language Runtime“ – zkráceně CLR) [51]. Tento modul zajišťuje běh aplikací a jejich kompilaci.

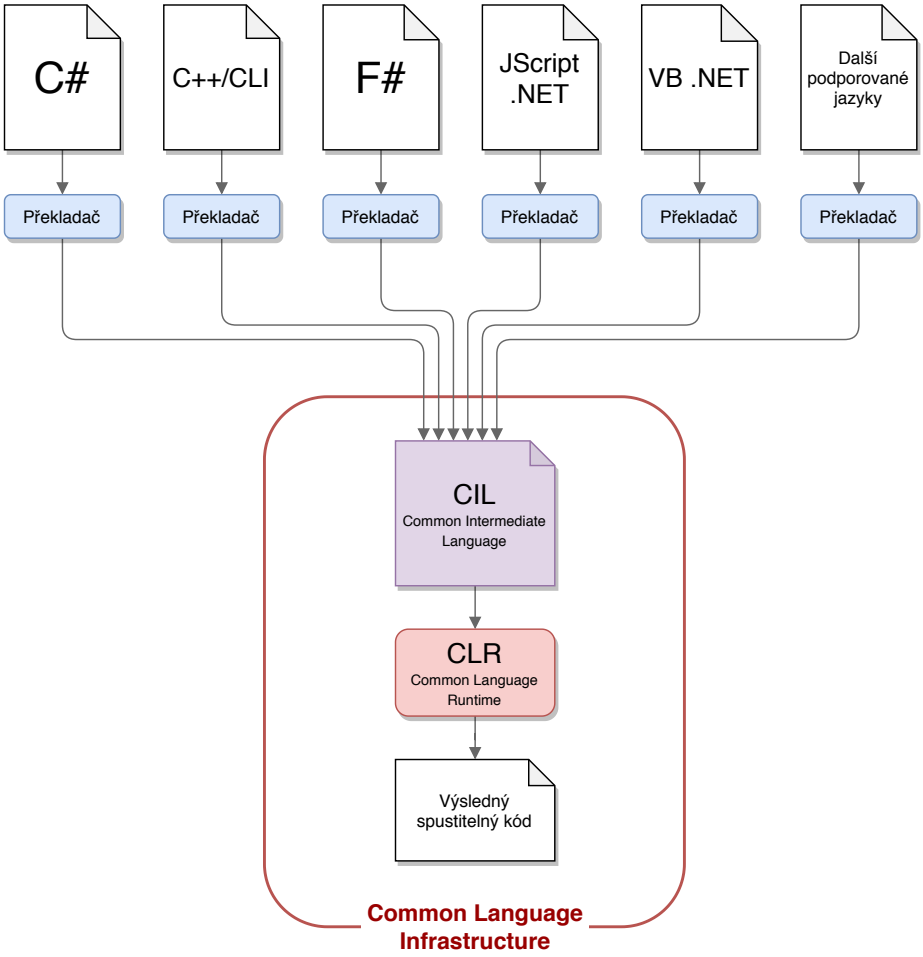
Při kompilaci aplikace vytvořené v .NET (a tedy využití běhového prostředí) zdrojové kódy nejsou překládány přímo do strojového kódu, který je možné rovnou provádět, ale do standardizovaného mezikódu CIL („Common Intermediate Language“, též používána zkratka MSIL – „Microsoft Intermediate Language“ nebo pouze IL – „Intermediate Language“) [15].

Tento postup se nazývá CLI („Common Language Infrastructure“) a je standardem uznaným mezinárodními společnostmi pro normalizaci ECMA [5] a ISO[6] [9]. Princip CLI je možné vidět na následujícím obrázku 3.8.

Důležitým stavebním blokem platformy .NET je podpora společného typového systému („Common Type System“, neboli CTS).

[5] Specifikace standardu ECMA-335 dostupná na: https://www.ecma-international.org/publications/standards/Ecma-335.htm

[6] Specifikace standardu [ISO/IEC 23271:2012] dostupná na: https://www.iso.org/standard/58046.html



Obrázek 3.8: Princip Common Language Infrastructure (zdroj: autor, data: [33])

Kompilátor (neboli překladač) tedy nepřekládá do nativního kódu, ale do CIL. Tento jazyk je podobný assembleru a je nezávislý na konkrétním procesoru. Oproti assembleru však nabízí více možností, umí například pracovat s objekty, zpracovávat výjimky nebo volat virtuální metody. Cílem při vývoji jazyka CIL byla snaha o zbavení se závislosti na platformě a možnost tak přenášet existující kód mezi více různými platformami ([8], kapitola 1.3).

K tomuto kódu jsou poté přidány datové soubory a je vytvořeno tzv. assembly. Výsledkem sestavy mohou být dva typy souboru, první variantou je proces sestavy `exe`, tou druhou je poté sestava knihovny s příponou `dll` [15].

Při spuštění assembly je provedena kompilace pouze potřebných částí programu, tyto části jsou přeloženy do strojového kódu optimalizovaného pro daný procesor a platformu. Toto má na svědomí tzv. „Just in time“ překladač (zkráceně JIT). Díky tomu, že se při spuštění nemusí překládat celá velká assembly, je proces spuštění rychlý [15].

Kód, jehož spuštění je řízeno běhovým prostředím, je nazýván jako řízený kód (anglicky „managed code“) [52]. U řízeného kódu je automaticky kontrolován korektní přístup do paměti a poskytuje prevenci pro přetečení vyrovnávací paměti. Výhodou je dále kromě zvýšené bezpečnosti aplikace také automatická správa paměti (anglicky „garbage collector“), kontrola datových typů za běhu programu a dozor nad ukazateli, zda ukazují na správný objekt [37].

Za nevýhodu by se dala považovat nemožnost přímo alokovat paměť a absence přístupu k nízké úrovni CPU architektury [37].

Historie a vývoj platformy .NET

První zmínka o .NET platformě se datuje ke konci 90. let minulého století, kdy společnost Microsoft začala vyvíjet .NET Framework pod názvem Next Generation Windows Services (NGWS).

Počáteční verze .NET frameworku spatřila světlo světa 13. února 2002 a značila se jako .NET Framework 1.0. Poprvé také vyšlo i vývojové prostředí Visual Studio .NET, současně ve stejném roce byl poprvé veřejnosti přístupný i programovací jazyk C# 1.0 a modul CLR 1.0 [58].

S příchodem platformy .NET Framework 2.0 roku 2005 došlo k nejvýraznějším změnám od vzniku aplikačního rámce, zejména na úrovni IL a v ASP.NET.

Technologie ASP.NET byla přepracována a přichází s inovovanou koncepcí pro vývoj, nasazení a samotný provoz webových aplikací [47]. Do jazyka IL společnost Microsoft implementovala nové prvky, mezi které patří generické typy, parciální třídy, statické metody a další.

Nutnou změnu zaznamenal i modul CLR, který se dočkal nové verze 2.0. Důvodem pro úpravu implementace byla nově představená funkcionalita v podobě generických typů, jejichž podporu bylo nutné přidat do běhového prostředí [2].

S verzí 3.0, která byla uvolněna 21. listopadu 2006, bylo možné ještě snadněji implementovat distribuované aplikace. Byly totiž vytvořeny nové komponenty, které jsou známy pod názvy:

- *Windows Presentation Foundation* (WPF)
- *Windows Communication Foundation* (WCF)
- *Windows Workflow Foundation* (WWF)

O rok později byla vydána verze .NET Framework 3.5 se kterou přišel i aktualizovaný C# 3.0. Bezpochyby největší novinkou této verze byla jednotná syntaxe pro přístup k datům, která byla v této verzi frameworku integrována. Tato syntaxe se nazývá LINQ („Language Integrated Query“) a přinesla revoluci v dotazování. První opravná aktualizace této verze zároveň přinesla další ulehčení při práci s daty, reprezentované rámcem Entity Framework (více v sekci 3.3.1) [58].

Další generace tohoto aplikačního rámce byla uvolněna na přelomu desetiletí s názvem .NET Framework 4.0. S jejím příchodem se představila i nová verze CLR 4.0 (značení 3.0 bylo u CLR přeskočeno a bylo zarovnáno s označením frameworku, v době psaní práce se zároveň jedná o nejnovější interpretaci tohoto modulu). Verze rámce s označením 4.5 vydaná v srpnu 2012 nově přinesla pro jazyky C# a VB .NET podporu pro asynchronní funkce užitím dvou klíčových slov `async` a `await` a podporu HTML5 pro rámec ASP.NET [58][2].

Obrovským milníkem ve vývoji platformy .NET byl rok 2015, kdy společnost Microsoft představila vizi s názvem .NET 2015. Jedná se o sadu nových technologií, které už byly ve finální fázi vývoje a společnost je v nejbližší budoucnosti plánovala zpřístupnit programátorské veřejnosti.

Hlavními komponentami .NET 2015 jsou [27][10][39]:

- **Aplikační rámce**
  - *.NET Framework 4.6* – minoritní aktualizace, která skýtala především opravy chyb a přidání podpory nových API
  - *.NET Core 1.0* – zatímco s platformou .NET Framework byli .NET vývojáři dobře srozuměni a doposud byla jedinou variantou pokrývající všechny scénáře, nyní jim přibyla možnost vybírat. Platforma .NET Core 1.0 (při představení označována dočasně jako .NET Core 5) je modulární open-source framework, který byl navržen pro vývoj aplikací s různým zaměřením, především však pro implementaci distribuovaných aplikací. Současně však s .NET Framework sdílí některé komponenty (.NET Core je blíže popsán v následující podkapitole).
- **Překladače**
  - *.NET Compiler Platform („Roslyn")* – kolekce open-source překladačů s důrazem na rozsáhlou analýzu kódu.
  - *RyuJIT* – nově přepracovaný 64-bitový JIT („just-in-time") kompilátor, který se stal implicitním JIT překladačem. Oproti původní verzi je značně vylepšena jeho výkonnost.
  - *.NET Native* – překladač, který kompiluje jazyky C# a Visual Basic do strojového kódu (podobně jako např. u jazyka C++)
- **Aplikační modely**
  - *ASP.NET Core 1.0* – od základů znovu postavený aplikační model pro tvorbu moderních webových aplikací (stejně jako v případě .NET Core 5 byl při představení použit pracovní název ASP.NET 5). Nyní se skládá z modulárních komponent a může být spuštěn nad běhovým prostředím .NET Core 5 (Core-CLR) a .NET Framework (CLR). Se zvoleným běhovým prostředím se liší i dostupné knihovny programátorovi. (Tento model je dále popsán v kapitole 3.3.1).

V současnosti nejnovější verzí .NET Frameworku je verze 4.8. Od chvíle představení platformy .NET Core se vývoj zaměřil spíše k tomuto rámci. Společně s druhou generací (a minoritními aktualizacemi) bylo přidáno velké množství knihoven a podpora snadné migrace webových aplikací na platformu .NET Core [20]. Verze 3.0 a 3.1 poté přinesly vylepšení výkonu, úspornější automatickou správu paměti, novou verzi jazyka C# 8.0 a dlouhodobou tříletou podporu (platí pouze pro verzi 3.1) [22].

.NET Core

Platforma .NET Core je novější open-source aplikační rámec, který se od zavedených standardů .NET liší především svou multiplatformitou. Není tedy závislý pouze na operačním systému Windows, ale přidává podporu i pro operační systémy macOS a Linux [20].

Společně s platformou .NET Framework sdílí přístup k API, které jsou definovány specifikací knihoven .NET Standard. Dále sdílí kompilery a podporu některých jazyků, jakými jsou C#, Visual Basic .NET a F# [42].

V porovnání s .NET Framework je navíc rychlejší, programátor má možnost si určovat pouze knihovny, které chce zahrnout a ty nevyužité nejsou ve výsledném projektu zařazeny [20]. Velké množství balíčků a knihoven třetích stran však doposud tato platforma nepodporuje. Poslední verzí je aktuálně .NET Core 3.1.

ASP.NET Core

ASP.NET Core je novou technologií pro tvorbu moderních webových aplikací a služeb užitím jazyka C#. Za jeho vývojem stojí společnost Microsoft, přesněji vznikla přestavbou architektury ASP.NET 4, a je tak součástí vývojářské platformy .NET. Jedná se o modulární framework, který je schopen fungovat v kombinaci s platformami .NET Core i .NET Framework. Tento fakt ovšem neplatí pro nejnovější verzi ASP.NET Core 3.0, která může být spuštěna pouze na platformě .NET Core [34].

Programátor může pro hostování aplikace volit z portfolia webových serverů, který kromě Microsoft IIS skýtá i populární server Apache, Nginx, Docker a další. Omezen není ani operačním systémem, na kterým aplikaci vyvíjí a spouští, podporovány jsou tři nejrozšířenější operační systémy: Windows, macOS a Linux [34].

Entity Framework Core

Entity framework Core je užitečný NuGet balíček (soubor knihoven) podporovaný firmou Microsoft. Jedná se o objektově-relační mapper, který umožňuje vývojářům využívat vyšší míru abstrakce, kdy programátor přistupuje pouze k .NET objektům, aniž by musel přistupovat přímo k databázovým tabulkám [12].

Programátorovi umožňuje při implementaci použít různé metody přístupů jak s realizací databáze naložit [40]:

- „The Database First“ – v doslovném překladu přístup „databáze první“ – tato metoda je pro vývojáře nejužitečnější v případě, že databáze již existuje a tak programátor stráví zlomek času vytvářením pouze vrstvy přistupující k datům (anglicky „Data Access Layer“, zkráceně „DAL“).
- „The Model First“ – přístup „model první“ využívá speciální sadu konceptů popisujících strukturu EDM („Entity Data Models“). V převážné většině případů je tato metoda používána jen v případě, chce-li vývojář při tvorbě databáze používat nástroj *Visual Entity Designer*. V opačném případě je doporučeno vybrat jeden ze dvou zbývajících postupů.
- „The Code First“ – poslední přístup umožňuje psát přímo kód aplikace jako první. Programátor vytváří aplikační logiku nad třídami, které reprezentují entity databáze. Databáze slouží jako „mechanismus uložení“ dat těchto tříd (nazývaných jako modely). Entity Framework Core poté databázi na základě těchto modelů vytvoří a reflektuje všechny změny v kódu, vývojář tak získá plnou kontrolu.

3.3.2 C#

C# je objektově-orientovaný programovací jazyk vytvořený a podporovaný společností Microsoft jako součást .NET Frameworku. V názvu lze vidět podobnost s jinými jazyky, to není náhodou, neboť C# vychází z programovacích jazyků C/C++, v mnohém je však podobnější jazyku Java [8].

C# podporuje jednoduchou dědičnost, zajišťuje typovou bezpečnost, obsluhu výjimek a další. Ani o správu paměti se programátor nemusí starat sám, pro tyto účely je zde používán takzvaný „garbage collector“ [8].

Užitím tohoto programovacího jazyka je možné implementovat širokou škálu aplikací s různými zaměřeními, mezi ně například patří nativní, mobilní a webové aplikace, hry, webové služby a další [48].

Velká část vlastností jazyka vyplývá přímo z platformy .NET, se kterou je C# úzce spjat. Proto i jeho vývoj v minulosti je nastíněn v historii platformy .NET (kapitola 3.3.1). Poslední majoritní verze jazyka byla představena v září 2019 a nese označení C# 8.

3.3.3 MySQL

Jedním z typů databáze je populární relační databázový systém MySQL. Je založen na deklarativním jazyku SQL („Structured Query Language“). MySQL nabízí podporu pro všechny hlavní platformy, jakými jsou Linux, UNIX a Windows a je šiřitelný zdarma. Disponuje zejména vysokým výkonem, rychlostí a vysokou kompatibilitou. Ačkoliv uvedený databázový systém je poměrně univerzální a může být použit ve velkém množství případů, je nejčastěji spojován s webovými aplikacemi [36].

3.4 Další použitý software

3.4.1 Git

Při vývoji softwarového projektu je užitečné uchovávat jednotlivé verze zdrojových souborů, seznam změn mezi verzemi a u větších projektů i autory změn. Takové systémy se nazývají verzovací systémy a je jím i zmiňovaný Git[7]. Umožňuje vývoj projektu rozdělit do jednotlivých vývojových větví, tzv. „branches“, kde každý vývojář, popřípadě tým vývojářů, pracují na určité části implementovaného projektu. Do hlavní vývojové větve (tzv. „master“) se zpravidla nahrávají pouze otestované a přeložitelné komponenty výsledného softwaru.

Verzovací systém je obvykle používán ve spojení se vzdáleným repozitářem, jedním z příkladů poskytovatele těchto webových služeb je Github.

3.4.2 XAMPP

Dalším softwarem použitým při vývoji byl softwarový balíček *XAMPP*[8]. Písmeno „X“ v názvu reprezentuje multiplatformitu aplikace a je tak dostupná pro operační systémy Windows, MacOS i Linux. V souhrnu se jedná o odlehčenou distribuci webového serveru Apache, jenž vývojářům umožňuje snadné vytvoření lokálního webového serveru za účelem testování [28]. Pro běh serveru využívá IPv4 adresu vyhrazenou pro localhost (`127.0.0.1`).

Základem balíčku jsou 4 komponenty: Apache, MariaDB (dříve MySQL), PHP a Perl. První a nejdůležitější komponenta Apache je samotným webovým serverem, který distribuuje obsah webu počítači.[28] Pro ukládání dat na webu je zapotřebí vytvoření a správa databáze, podporovány jsou relační databáze MariaDB (vychází z MySQL – je zachována binární kompatibilita databázových souborů s poslední verzí MySQL, ve většině ohledů je chování MariaDB a MySQL identické a je tedy zajištěna i podpora MySQL databází)[25]. Dvě „P“ na konci názvu značí podporu interpretovaných skriptovacích jazyků PHP a Perl.

[7] Dostupné na: https://git-scm.com
[8] Dostupné na: https://www.apachefriends.org

Mimo to *XAMPP* nabízí další užitečné nástroje, mezi které se řadí *Filezilla FTP server* pro manipulaci se soubory a uživatelské rozhraní pro práci s databází *phpmyadmin* [28].

3.4.3 Postman

Aby bylo možné otestovat webové API bez nutnosti implementace klienta, vznikl software jménem *Postman*[9]. Tato aplikace dokáže suplovat přítomnost klienta a v jeho roli umožňuje posílání HTTP požadavků tak, jako by je inicioval právě klient. Odpovědi ze strany serveru je aplikace schopna zobrazit v různých formátech. Jednotlivé dotazy lze ukládat a snadno organizovat do složek v přehledném uživatelském prostředí [45].

Kromě interakce se serverovým API lze v aplikaci vytvářet automatizované testy, monitoring, dokumentaci a další [45].

Software je dostupný na operační systém Windows, MacOS i Linux.

[9] Dostupné na: https://www.postman.com

Kapitola 4

Návrh systému

Jak již bylo zmíněno v kapitole 2, mým cílem bylo vytvořit webovou aplikaci, ve které je pracováno s rodokmeny. Snahou bylo navrhnout komplexní aplikaci, jež by byla schopna konkurovat existujícím řešením popsaným v kapitole 2.3 a současně si zachovat jednoduchý nekomplikovaný vzhled. Rozmístění některých elementů bylo inspirováno aplikacemi řešícími obdobnou problematiku. Návrhu samotného uživatelského prostředí je věnována podkapitola 4.3, předchází jí však text věnovaný návrhu databáze a samotnému členění aplikace. Navržená databáze musí nutně reflektovat všechny požadavky zmíněné v sekci 2.2, současně však jsem se snažil zachovat její přehlednost a uchovávat ve většině případů pouze nezbytná data. Dalším cílem bylo zefektivnit přístupy do databáze a minimalizovat velikost přenesených dat po síti.

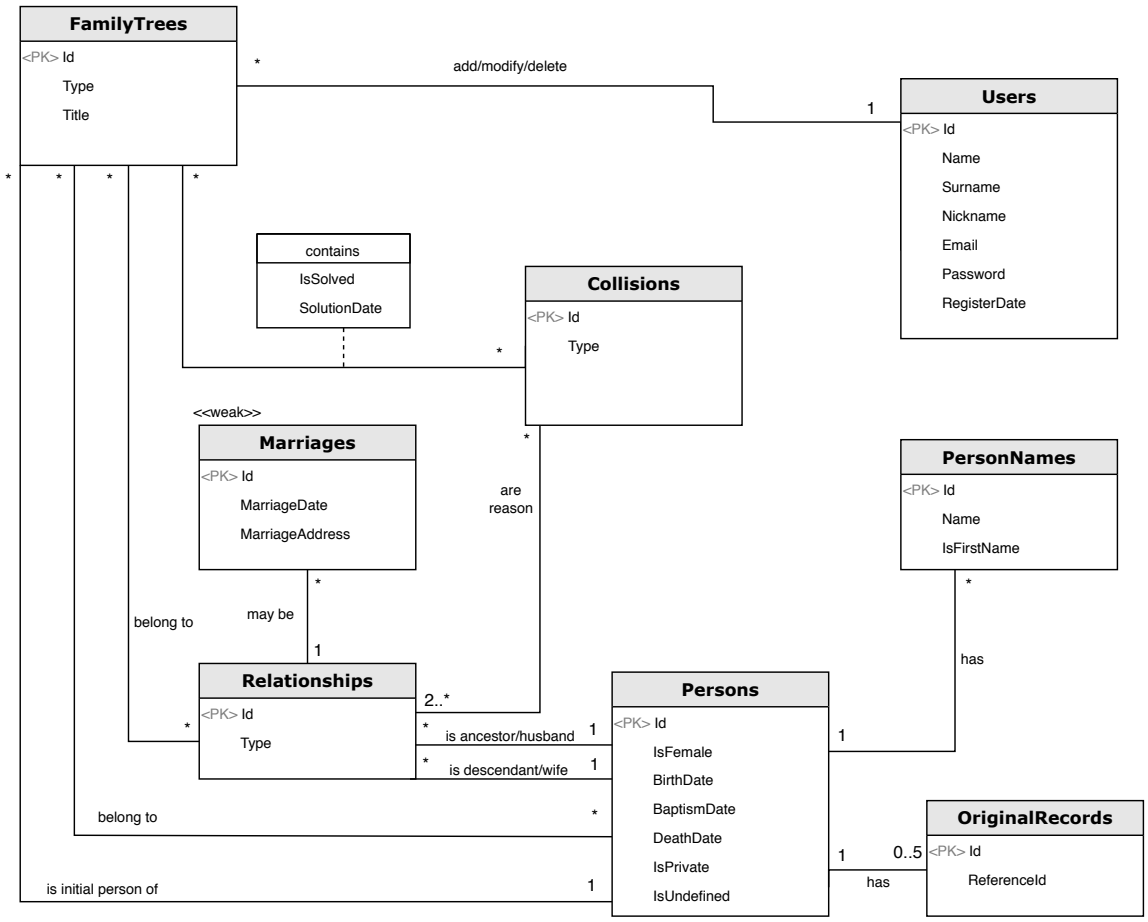
4.1 Návrh databáze

Žádná moderní webová aplikace se neobejde bez databáze. První složkou implementovaného systému je tak relační databázová vrstva. Jedním ze způsobů jak popsat návrh databáze s určitou mírou abstrakce je jazyk UML (zkratka z „Unified Modeling Language“). Užitím UML notace je možné vizualizovat a popsat způsoby uložení dat. Příkladem takového popisu může být i stavový model zvaný ER diagram („Entity Relationship Diagram“).

ER diagram je definován následujícími základními termíny:

- *Entita* – Entita reprezentuje jakýkoliv objekt reálného světa, který však musí splňovat podmínku rozlišitelnosti od ostatních objektů. Příkladem by mohl být uživatel s identifikátorem 22.
- *Entitní množina* – Množina entit, identického typu, které sdílí stejné vlastnosti (atributy). Entitní množiny jsou zaneseny v ER diagramu. Ukázkou je například *Osoba*, *Rodokmen*, *Kolize*, ...
- *Atribut* – Jedná se o konkrétní vlastnost entity. Atributy jsou jméno uživatele, příjmení a podobně.
- *Vztah* – Vztahem se rozumí asociace mezi entitami, například: uživatel s identifikátorem 91 je autorem rodokmenu s identifikátorem 202.
- *Vztahová množina* – Jde o množinu vztahů, které mají shodné vlastnosti. Jako ukázka poslouží vztahová množina: vztahy patří do rodokmenů.

Na obrázku 4.1 lze vidět vytvořený kompletní ER diagram, který je navržený tak, aby splňoval požadavky na aplikaci uvedené v kapitole 2.2. Jednotlivé tabulky databáze a jejich význam je popsán níže, stejně tak význam jejich atributů a vztahů. Všechny tabulky disponují generovaným identifikátorem, který je v rámci tabulky unikátní. Vykreslený ER diagram je záměrně v anglickém jazyce, aby co nejvíce korespondoval se zdrojovými kódy vyvinuté aplikace.



Obrázek 4.1: Návrh systému – ER diagram

Tabulka Users

Tato tabulka reprezentuje uživatele registrovaného v aplikaci a uchovává o něm všechny zadané informace.

Primárním klíčem tabulky je *id*, dále jsou uchovávány informace o jeho jméně, příjmení, emailu, uživatelském jméně a hesle. Při registraci je uživatel povinen zadat pouze uživatelské jméno a heslo, neb se jedná o údaje, pomocí kterých se do aplikace přihlašuje, vyplnění ostatních údajů je volitelné. Protože uživatelské jméno slouží pro jeho identifikaci při procesu přihlašování, musí být v rámci aplikace unikátní.

Dále je uchována informace o datu registrace, ta však slouží pouze jako metadata, která jsou uživateli prezentována pouze pro zajímavost.

Každý uživatel může vytvářet a upravovat více rodokmenů, současně však je autorem pouze on a nikdo jiný není oprávněn je upravovat.

Tabulka FamilyTrees

V tabulce *FamilyTrees* jsou uchovávány základní informace o rodokmenech. Primárním klíčem tabulky je identifikátor *id*, mimo to je uložen typ rodokmenu (možnosti popsané v kapitole 2.2) v podobě výčtového typu a uživatelem zadaný název.

V databázi jsou udržovány informace o tom, které vztahy patří do rodokmenu a která osoba je zakladatelem rodu. Tyto vztahy jsou nezbytné pro zkonstruování a vykreslení rodokmenu (o této problematice je hovořeno až později, v kapitole 5.3). Vztah mezi rodokmeny a osobami slouží například při zjišťování, ve kterých veřejných rodokmenech se osoba nachází.

Tabulka Persons

Tabulka *Persons* je předlohou pro ukládání dat o jednotlivých osobách v rodokmenu.

Primárním klíčem tabulky je *id*, dále se uchovávají informace o pohlaví, místě narození a datu narození, křtu a úmrtí. Atribut *IsPrivate* je odvozený od typu rodokmenu, je datového typu *boolean* (pravdivostní hodnota) a značí, jak bude nakládáno s osobou v aplikační logice. Pokud je totiž osoba označena jako privátní, není brána v úvahu při vyhledávání podobných osob nebo při detekci kolizí.

Atribut *IsUndefined* slouží k selekci aplikací „uměle vytvořených“ osob, které byly založeny pro potřeby vykreslování. Použitá knihovna pro vykreslování rodokmenů `dTree` (3.2.5) vyžaduje přítomnost obou předků. Když tedy uživatel vytváří například otce, musí být založena i matka. Vytvoří se tedy „nedefinovaná osoba“, která může být později nahrazena. Současně může uživatel nad nedefinovanou osobou dále vytvářet rodokmen a v případě, že tedy uživatel nezná konkrétní osobu, ale zná její předky, nebrání mu tato skutečnost ve vytváření rodokmenu.

Tabulka PersonNames

V této tabulce jsou uchovávána jména pro danou osobu v rodokmenu.

Primárním klíčem tabulky je identifikátor *id*, kromě něj je uchováváno samotné jméno v podobě řetězce znaků a pravdivostní hodnota, zda se jedná o křestní jméno či nikoliv. Pokud je hodnota posledního atributu nepravdivá, je implicitně předpokládáno, že se jedná o příjmení.

Potřeba pro tuto tabulku vznikla s úvahou, že každá osoba může mít více jmen, například při sňatku nevěsta (či ženich) přijme příjmení svého partnera. Současně však může být v databázi vyhledán podle všech nabytých příjmení.

Tabulka Relationships

Tabulka *Relationships* představuje vztah mezi dvěmi osobami. Jejím primárním klíčem je *id*. Mimo něj uchovává pouze typ vztahu, který je výčtového typu. Může nabývat hodnot:

- isMotherOf (je matkou)
- isFatherOf (je otcem)
- isInMarriageWith (je v manželství)

Při implementaci byl řešen problém jak rozlišit účastníky příbuzenského vztahu, ku příkladu kdo ze dvou osob je matka a kdo potomek. Tento problém vyústil v řešení založením dvou různých vztahů mezi osobou a vztahem:

- předek/manžel
- potomek/manželka

Na základě participace osoby v jednom z těchto dvou vztahů lze snadno rozpoznat její pozici ve vztahu.

Tabulka Marriages

Tabulka *Marriages* je v ER diagramu (obrázek 4.1) označena jako slabá entitní množina, neboť je existenčně závislá na tabulce vztahů. V případě, že je vztah typu „manželství“, doplňuje informace k tomuto vztahu datem a místem svatby. Pokud dojde k zániku příslušného vztahu, v databázi zaniká i doplňující záznam o svatbě. Ke každé svatbě však může existovat více doplňujících záznamů o svatbě, protože participující osoby mohly uzavřít sňatek vícekrát.

Tabulka Collisions

V tabulce *Collisions* je uchováván pouze typ kolize a identifikátor *id*, který je současně primárním klíčem. Typ kolize je výčtového typu a může nabývat hodnot:

- differentMother (rozdílná matka)
- differentFather (rozdílný otec)
- marriageOrAncestor (manželství nebo předek)

Příčiny konfliktu, který tyto typy vyjadřují, jsou blíže popsány v kapitole 2.2, přičemž první dva uvedené typy odpovídají příčině typu „různý předek“, zbývající typ je poté shodný s typem „kolize generací“.

Důležitou součástí kolize je vztah s tabulkou *Relationships*, kde jsou to právě záznamy této tabulky, na kterých je záznam kolize existenčně závislý. Aby mohl záznam o kolizi existovat, musí být navázaný na minimálně dva záznamy tabulky *Relationships*.

Dále je pak uchovávána informace o tom, kterých rodokmenů se kolize dotýká. Vztah s tabulkou *FamilyTree* obsahuje atributy a je tak povýšen na entitu. Je rozšířen o informace související s vyřešením kolize pro daný rodokmen. Těmito informacemi se rozumí pravdivostní hodnota o vyřešení *isSolved* a datum, kdy autor rodokmenu konflikt řešil – atribut *SolutionDate*.

Tabulka OriginalRecords

Tato tabulka řeší problematiku ukládání referencí na matriční záznamy osoby. Jsou v ní uchovávány záznamy o identifikátorech *ReferenceId*, které mají význam pouze v databázi digitalizovaných matrik, na kterou se tato práce odkazuje.

V této databázi se osoby vyskytují v různých záznamech pod odlišnými identifikátory. Provázání osoby mezi implementovanou aplikací a databází digitalizovaných matričních záznamů probíhá skrze identifikátor záznamu o narození osoby, který je uložen v odkazované databázi.

Zmíněný identifikátor je v aplikaci potřeba uložit. Vznikla dvě možná řešení, jak reference uchovat:

1. *Uchovat jediný identifikátor* – pro každou osobu by se dalo odkázat na původní záznamy pomocí jediného identifikátoru. V tabulce *Persons* by existoval další atribut, který by pro každý záznam nesl hodnotu odkazu. Na straně originální databáze by existovala logika, která k jednomu určitému identifikátoru přiřadí všechny odpovídající identifikátory v databázi matrik a záznamy zobrazí.

2. *Uchovat všechny identifikátory* – v databázi by bylo nutné vytvořit novou tabulku, která by obsahovala jediný atribut – identifikátor na jeden z původních záznamů. Tato tabulka by byla ve vztahu 1:N k tabulce *Person*. Každá osoba by tak byla propojena s N záznamy v původní databázi.

Po konzultaci s vedoucím práce a současně spoluautorem odkazovaného projektu byla vybrána druhá možnost, a proto byla založena tabulka *OriginalRecords*. Počet maximálních záznamů však byl omezen na pět, více získaných identifikátorů by už znamenalo přílišnou neurčitost.

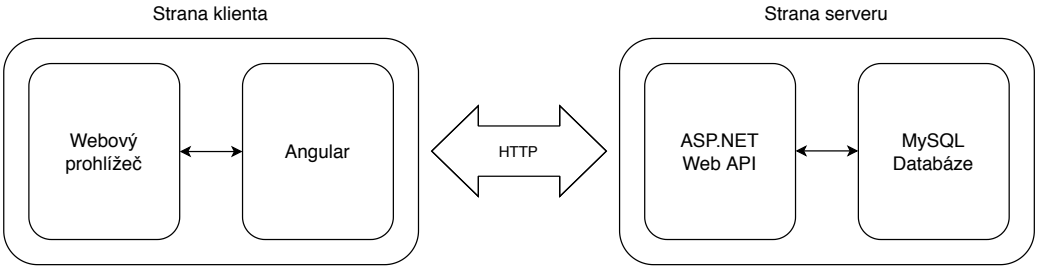
4.2 Návrh architektury aplikace

Vytvářený systém se skládá ze tří hlavních komponent. První komponenta byla již zmíněna a blíže popsána v předchozí kapitole a je jí samotná databáze pro uchovávání dat. Tato databáze využívá relační databázový systém MySQL.

Rozvržení dílčích komponent je navrženo, aby implementovalo model třívrstvé architektury popsané v kapitole 3.1.2. Aplikace je tak strukturována do tří vrstev, kde každá zastává svoji specifickou roli:

- databáze (datová vrstva)
- webové API (aplikační vrstva)
- angularová aplikace (prezentační vrstva)

Na obrázku 4.2 je vyjádřena struktura navrhovaného systému. Ze schématu je patrné, že navržený systém rovněž koresponduje s architekturou klient–server (3.1.1) a je tak rozdělen na aplikační logiku s databází spuštěnou na serveru a část, která přímo interaguje s prohlížečem na straně klienta.



Obrázek 4.2: Návrh systému – Schéma implementované aplikace

Aplikační logika (implementována na straně serveru) je řešena jako program disponující takzvaným RESTful API. Tento název je terminologií a neexistuje pro něj adekvátní český překlad. Představuje aplikační rozhraní, na kterém program přijímá HTTP žádosti následujících typů: GET, POST, PUT a DELETE. Aplikace mimo těchto služeb zajišťuje výpočty, provádí dotazování nad implementovanou databází a vykonává další operace. Na základě žádostí uživatele načítá data z databáze, zpracovává je, vyhodnocuje a formátovaná data posílá klientovi. Toto Webové API je implementováno užitím platformy ASP.NET Core (3.3.1) a jazyka C# (3.3.2).

Komunikace mezi aplikací angularu a webovým API probíhá pomocí HTTP žádostí, přičemž ji iniciuje klient. Webová aplikace běžící na serveru takovéto žádosti zachytává a předává je ke zpracování kontrolerům. Danému kontroleru je žádost přiřazena na základě jejího typu a URL adresy. Server obratem klientovi odesílá HTTP odpovědi.

Aplikace na straně klienta je implementovaná webovým rámcem Angular (3.2.4). Formátuje vzhled webové stránky, který je distribuován uživateli a zpracovává uživatelovu interakci s webovou stránkou. Současně aplikace kontroluje validitu uživatelských vstupů a žádosti, které uživatel manipulací se stránkou generuje, odesílá na server.

Uživateli je v grafickém uživatelském rozhraní poskytnuta možnost přesměrování na záznamy v databázi digitalizovaných matrik. Cíl přesměrování je závislý na tom, zda jsou pro osobu v aplikaci přiřazeny konkrétní záznamy:

- osoba má přiřazené matriční záznamy – aplikace k odkazu připojí identifikátor a uživatele přesměruje na stránku se zobrazeným původním záznamem
- osoba nemá přiřazené matriční záznamy – uživatel je přesměrován na seznam možných záznamů, které odpovídají zadaným údajům osoby

4.3 Návrh uživatelského rozhraní

Primárním účelem implementovaného systému je vytváření rodokmenů, a proto se grafický návrh uživatelského rozhraní snaží o efektivní a jednoduché zprostředkování této zkušenosti. Některé prvky řešení byly inspirovány již existujícími aplikacemi popsanými v kapitole 2.3.

Vzhled aplikace je navržen způsobem, aby byl přehledný, líbivý a současně dostatečně kontrastní. Je využívána koncepce moderního designového jazyka *Material Design*, který sází na jednoduchost prvků. Pro webový rámec Angular existují za tímto účelem komponenty Angular Material[1], jenž jsou v projektu použity. Responzivní rozložení prvků na stránce je zajištěno užitím specializovaného API Angular Flex-Layout[2].

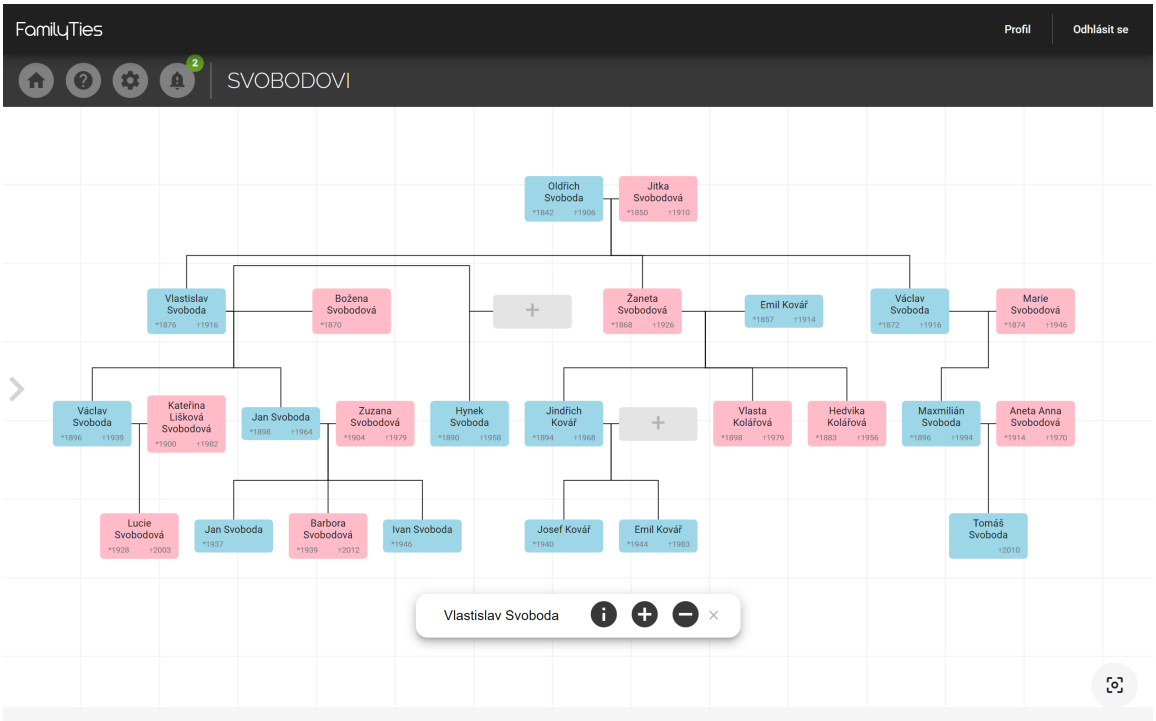
[1] Více o Angular Material na: https://material.angular.io/
[2] Více k API Angular Flex-Layout na: https://github.com/angular/flex-layout

V práci je nejdůležitější částí manipulace s rodokmenem, na kterou je směřována většina pozornosti. Na obrázku 4.3 je možné vidět návrh zobrazení, které se naskytne uživateli při vytváření rodokmenu. V tomto případě již má uživatel jistou část rodokmenu vytvořenou. Rozvržení ovládacích prvků v této části aplikace je rozděleno do dvou segmentů.

První část tvoří tmavě šedý panel umístěný v horní části obrazovky. Jeho obsah je zarovnán vlevo, přičemž je oddělen svislou čarou. Nalevo od dělící čáry je možné vidět čtyři funkční tlačítka, jejichž akce operují nad rodokmenem jako celkem. Nejdůležitějším tlačítkem v této sekci je tlačítko se symbolem zvonu s vykřičníkem. Kliknutím na tuto ikonu si uživatel může zobrazit konflikty existující pro daný rodokmen. Současně v pravém horním rohu tlačítka je zobrazen v malé „bublině“ celkový počet dosud nevyřešených kolizí. Tlačítko kolizí není přítomné v privátních rodokmenech, takovéto rodokmeny totiž testům na konflikty nejsou podrobeny (2.2). Napravo od dělící čáry je hůlkovým písmem zobrazen název rodu.

Druhým segmentem jsou ovládací prvky umístěné ve spodní části obrazovky. Tento blok skýtá tlačítka pro operace s konkrétní osobou v rodokmenu. Je uživateli ukázán pouze v situaci, kdy je vybrána osoba. Výběr lze provést kliknutím na bublinu se jménem příslušné osoby.

Většina tlačítek v tomto zobrazení je pro jednoduchost vzhledu reprezentována pouze ikonami, při najetí myší se však uživateli zobrazí text s popisem akce, kterou dané tlačítko zastupuje.



Obrázek 4.3: Grafický návrh uživatelského rozhraní – zobrazení rodokmenu jeho autorem

Na obrázku 4.3 je dále možné vidět, že bubliny se jmény osob jsou barevně rozlišeny. Je tomu učiněno v závislosti na pohlaví jedince, kdy modrá barva reprezentuje muže a růžová ženy. Šedou barvou s ikonou + jsou označeny neznámé (nedefinované) osoby. Rok narození a úmrtí je v bublině vyobrazen pouze v případě, že je uživatelem zadán.

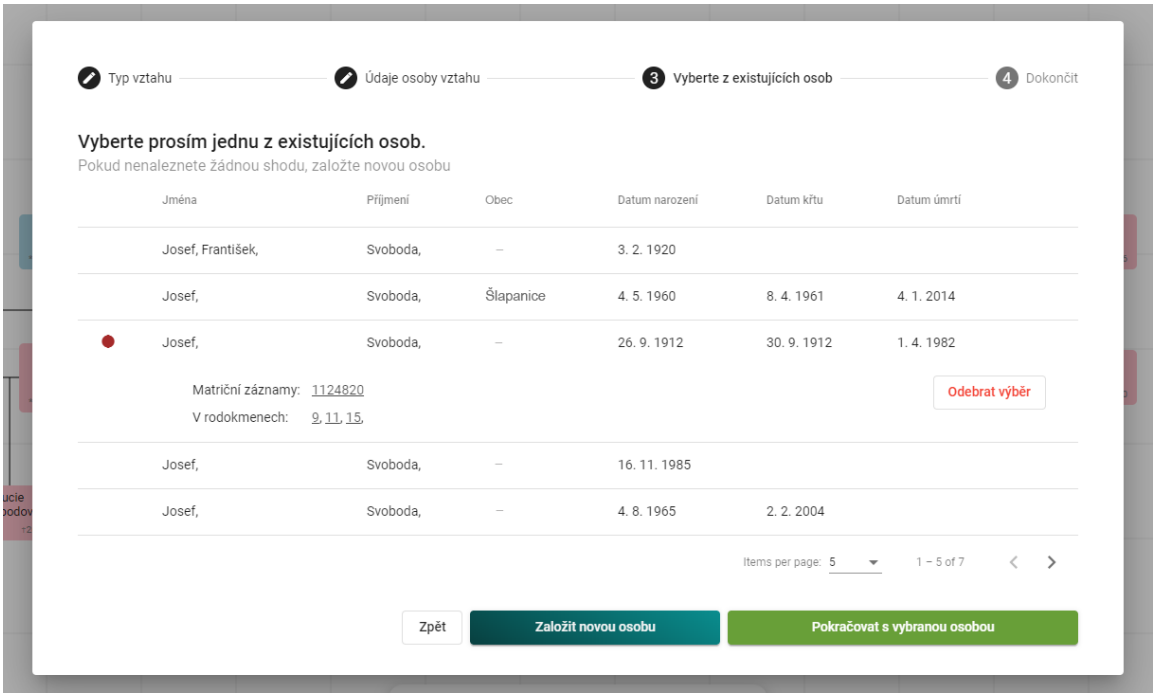
S rodokmenem je možné pohybovat do všech stran a různě ho přibližovat či oddalovat. Pokud jej chce uživatel uvést do výchozí pozice, může k tomu využít tlačítko umístěné v pravém dolním rohu. Mimo to je rodokmen automaticky překreslen při změně velikosti okna webového prohlížeče.

Při pohledu na obrázek 4.3 je patrné, že pro každou osobu je v rodokmenu vyobrazen pouze zlomek informací. K zobrazení více informací o osobě návrh počítá s panelem, který je po akci uživatele vysunut z levé strany. Tento prvek byl inspirován konkurenčním systémem *MyHeritage* (2.3.1). Panel zabírá jednu čtvrtinu šíře okna prohlížeče a poskytuje detailní informace o osobě (zadaná data, rodiče, manželství, ...). Dále zpřístupňuje uživateli možnost přesměrování na digitalizované matriční záznamy.

Ve zbylé části obrazovky je i nadále zobrazován rodokmen a uživatel s ním může bez omezení pracovat. Za předpokladu, že uživatel ponechá otevřenou kartu s podrobnými informacemi a vybere jinou osobu, obsah okna je pro vybranou osobu aktualizován.

Akce, které uživatel svou interakcí s webovou stránkou vyvolává, jsou v hojném počtu případů navrženy jako modální okna. Příkladem může být seznam konfliktů nebo proces přidávání nové osoby (obrázek 4.4). Uživatel tak má stále pocit, že neopustil aplikaci úpravy konkrétního rodokmenu.

Zakládání nové osoby v rodokmenu je sekvenční proces čtyř kroků. Nejzajímavější částí je třetí krok, kde uživatel může vybírat z již existujících osob v databázi. Grafický návrh této části je možné vidět na obrázku 4.4.



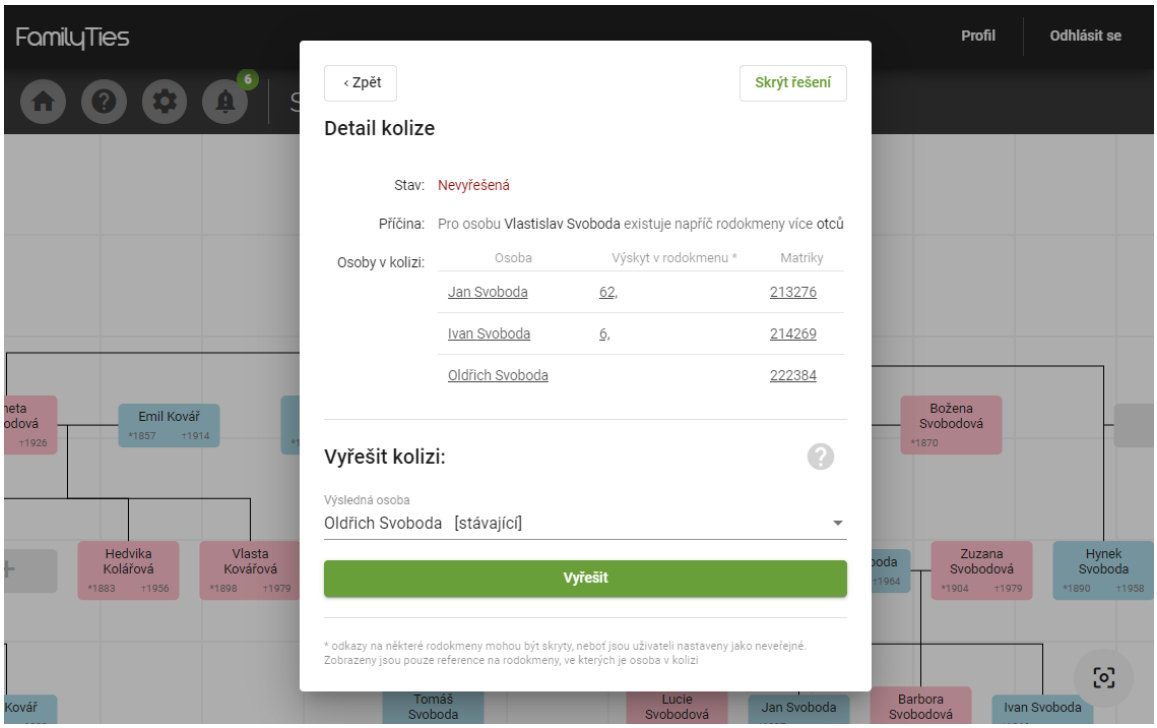
Obrázek 4.4: Grafický návrh uživatelského rozhraní – proces výběru podobné osoby

Alternativní osoby jsou uživateli zobrazeny v jednoduché přehledné tabulce. Kliknutím na záhlaví sloupce tabulky je možné záznamy vzestupně nebo sestupně řadit. Ve spodní části modálního okna může uživatel vybírat počet zobrazených záznamů a případně se přesouvat mezi stránkami tabulky.

Na výše uvedeném obrázku je pro jeden ze záznamů zobrazena karta s detailnějšími informacemi, kterou lze rozevřít a opět uzavřít kliknutím na konkrétní řádek. Karta nabízí přehled rodokmenů, ve kterých se dotyčná osoba vyskytuje, a odkazy na digitalizované matriční záznamy. Právě tyto záznamy mohou uživateli pomoci při rozhodování, která z nabídnutých osob je ta správná pro jeho rodokmen.

Výběr určené osoby je možné provést tlačítkem na pravé straně zobrazené karty. Pokud je existující osoba vybrána, záznam je označen červeným bodem na začátku řádku.

Dalším zajímavým bodem grafického uživatelského rozhraní je detail konkrétního konfliktu.

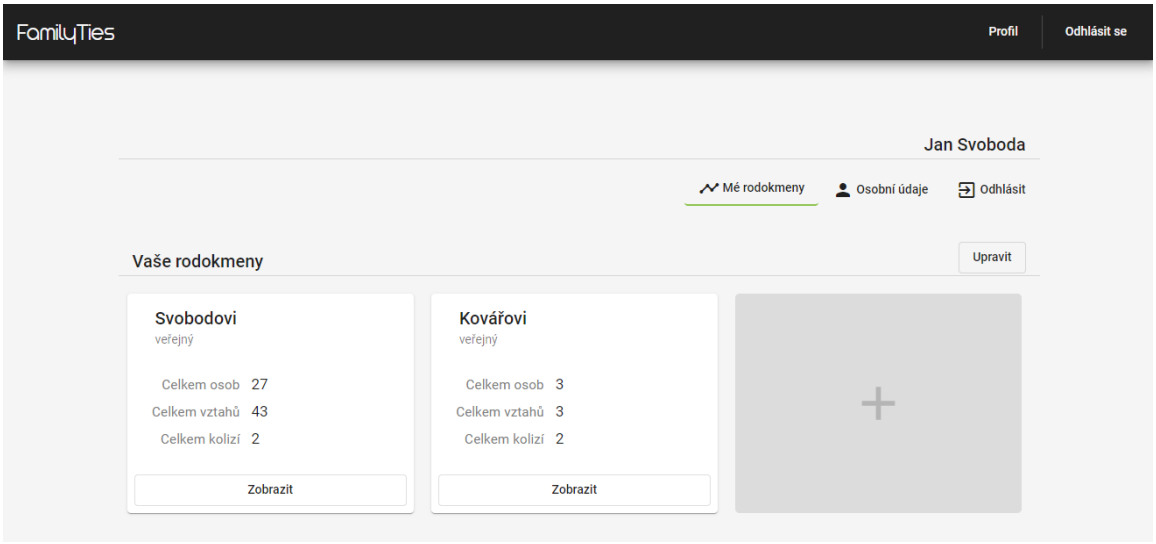


Obrázek 4.5: Grafický návrh uživatelského rozhraní – proces výběru podobné osoby

Když si uživatel prohlíží seznam všech kolizí v rodokmenu, má možnost si ke každé z nich zobrazit podrobnosti. V takovém případě je mu zobrazen detail příslušného konfliktu.

Na obrázku 4.5 jsou k vidění informace ke kolizi, která doposud nebyla vyřešena. Možnost řešení kolize je aktivována kliknutím na tlačítko v pravém horním rohu modálního okna. Na uvedeném obrázku je proces řešení již aktivován a je zobrazen pod horizontální dělící čarou. Uživatelské rozhraní řešiteli kolize nabízí seznam osob, jenž mohou být řešením konfliktu.

Každý uživatel si může vytvářet nekonečně mnoho vlastních rodokmenů. Přehled svých rodokmenů má k náhledu na svém profilu včetně možnosti vytvořit nový. Návrh grafického rozhraní pro tento přehled je k vidění na obrázku 4.6.



Obrázek 4.6: Grafický návrh uživatelského rozhraní – profil uživatele

Na profilu má uživatel možnost vybírat ze záložek pro zobrazení vlastních rodokmenů nebo svých osobních informací. Uživatelovy rodokmeny jsou zobrazeny v podobě jednotlivých karet se základními informacemi. Pod panelem se záložkami na obrázku 4.6 se nachází tlačítko „Upravit“, které má rozdílnou funkci v závislosti na zvolené záložce. V zobrazení rodokmenů se jeho stiskem zpřístupní úprava či smazání rodokmenu, v záložce osobních údajů má možnost tyto soukromé informace upravit.

Pro aplikaci bylo navrženo vlastní logo (obrázek 4.7). Samotné obrázkové logo je možné vidět pouze na úvodní obrazovce implementovaného systému. Rodokmen bývá lidově pojmenován jako „strom života“ a proto logo nese podobu stromu. V popředí zobrazuje dvě osoby, které splývají v kořeny. Toto spojení symbolizuje zakladatele rodu a jejich následníky.

Obrázek 4.7: Grafický návrh uživatelského rozhraní – logo aplikace

Název *FamilyTies* se skládá ze dvou slov, které v překladu z anglického jazyka znamenají „rodinné vazby“. Bylo tak zamýšleno vyjádřit hlavní smysl aplikace, kterým je vzájemně mezi sebou propojovat příbuzné osoby a zjišťovat o nich více informací.

Kapitola 5

Implementace

Tato kapitola si klade za cíl podrobně popsat důležité body a postupy použité při implementaci systému pro správu rodokmenů. Výsledný projekt je řešen jako webová aplikace, která i použitými technologiemi reflektuje model třívrstvé architektury popsaný v úvodu práce (kapitola 3.1.2). Samotné abstraktnější rozdělení aplikace je popsáno v předcházející sekci 4.2, kde jsou nastíněny i technologie použité při realizaci projektu. Tato kapitola na ni přímo navazuje a podrobněji popisuje řešení problematiky z pohledu implementace.

Pro správné pochopení je nejprve potřeba vysvětlit terminologii v technologii .NET, která využívá koncept *řešení* a *projektů*. Všechny soubory v rámci projektu jsou kompilovány do spustitelného souboru, řešení (anglicky „solution“) je poté kontejnerem pro jeden nebo více souvisejících projektů. Je zvykem rozdělovat logické bloky aplikace do jednotlivých projektů, čímž se řídí i implementace této práce.

5.1 Datová vrstva

Datová vrstva aplikace reprezentuje systém pro perzistentní ukládání dat. Uchování informací v projektu je řešeno použitím MySQL databáze, při implementaci projektu však byla použita metoda „Code First“ (3.3.1), která programátora odstíní od nutnosti pracovat s jazykem MySQL.

Při implementaci byly všechny entitní množiny z ER diagramu (obrázek 4.1) převedeny a transformovány na třídy v jazyce C# .

Při použití tohoto přístupu je totiž každá tabulka databáze realizována ve zdrojovém kódu jako třída (někdy nazývaná jako model). Vlastnosti třídy (anglicky „properties“) představují jednotlivé sloupce tabulky v databázi a jsou identické s atributy entitních množin ve zmiňovaném ER diagramu.

Současně jsou zachovány výhody jazyka C#, mezi které patří dědičnost a lze ji tak snadno promítnout i do implementovaných modelů. V projektu je tato funkce jazyka demonstrována na třídě `BaseEntity` implementující jedinou vlastnost, kterou je číselný identifikátor `id`. Všechny modely, které reprezentují entitní množinu, z této třídy dědí, díky čemuž zaniká nutnost implementovat tento stejný prvek pokaždé znovu.

Relace mezi entitními množinami jsou převedeny do podoby primárních a cizích klíčů v jednotlivých třídách. Pro vztahy N:N musely být zavedeny nové třídy reprezentující tyto vztahy, uvedené třídy jsou umístěny ve složce `Relationships`. Případné atributy vztahu jsou součástí konkrétní třídy tohoto typu. V uvedeném formátu se však jedná pouze o čí-

selné hodnoty a kolekce objektů, pro správné fungování vztahů mezi tabulkami je potřeba vlastnosti vzájemně namapovat.

K vzájemnému propojení tříd a vytvoření relací je třeba instance třídy `ModelBuilder`, která konfiguruje vytváření relací a mapování entit na tabulky databáze. Specifikace vztahů je provedena v třídě `ModelBuilderRelationships`.

Z nakonfigurovaných modelů je vygenerována MySQL databáze. Aby bylo možné nad databází provádět dotazy a všeobecně manipulovat s daty, je nutné vytvořit tzv. kontext. Ten je možný vytvořit derivací ze třídy `DbContext` a v této práci je implementován třídou `FamilyTreeDbContext`. Veškeré entitní modely jsou uloženy v projektu `Entity` a konfigurace kontextu a relací v projektu `Service`.

Zmíněnou funkcionalitu, kterou zahrnuje přístup „Code First“, přidává NuGet balíček Entity Framework Core (sekce 3.3.1), který je do řešení importován.

5.2 Aplikační vrstva

Jak už bylo zmíněno v kapitole 4.2, aplikační logika vyvíjeného systému je realizována jako Webové API. Implementovaný projekt tedy musí být schopen zachytávat a zpracovávat HTTP požadavky, této role se v ASP.NET Core ujímají takzvané kontrolery.

Pro každý model reprezentující tabulku databáze je vytvořen kontroler nesoucí zodpovědnost za odpovídání na žádosti klienta přicházející z webové stránky. Jejich pojmenování je určeno konkatenací názvu modelu s příponou „Controller“, přičemž obsluhují žádosti příslušící k danému modelu.

Ku příkladu kontroler s názvem třídy `UserController` zajišťuje manipulaci s daty tabulky `User` a zpracovává všechny požadavky odeslané na adresu `api/user`, kde zkratka `api` představuje URL adresu webové stránky. Jednotlivé metody kontroleru definují typ zpracované HTTP žádosti a URL adresu, pro kterou metoda žádosti zachytává. Všechny třídy zastávající funkci kontrolerů jsou hromadně umístěny ve složce „Controllers“ projektu `family-tree`.

Odesílaná data jsou převedena a přenášena ve formátu pro výměnu dat JSON. Za účelem odstínění struktury databázových tabulek a přenášených dat jsou vytvořeny modely, které obsahují pouze vlastnosti, jež je zapotřebí přenášet. Tyto modely jsou uloženy v projektu `PresentationModels` a v názvu disponují příponou `Dto` reprezentující anglické „Data transfer object“, neboli objekt pro přenos dat.

Ve velkém počtu případů jsou shodné s modely reprezentujícími tabulky relační databáze, umožňují však upravovat přenášená data bez nutnosti zasahovat přímo do databázových tabulek. Tato funkcionalita je užitečná například při přenosu údajů o uživateli (modelem `UserDto`), kdy data jsou přenášena bez zašifrovaného hesla a kryptografické soli navzdory tomu, že databázový model `User` tyto vlastnosti obsahuje. Naopak při přihlašování uživatele jsou přenášeny pouze údaje o uživatelském jméně a hesle bez nutnosti posílat (v tomto případě prázdné) další vlastnosti.

Metody kontroleru pouze zachytávají žádosti a volají jejich obsluhu. Řízení v takové chvíli předávají metodě implementované v některé ze služeb (anglicky „service“), která v zápětí požadované akce provede. Třídy služeb jsou podrobněji popsány níže, důležitý v této chvíli je fakt, že na vstupu očekávají databázový model nebo hodnotový datový typ, nikoliv model pro přenos dat. Protože `Dto` objekty obsahují stejné vlastnosti jako databázové modely, pouze jinak strukturované, je možné jednotlivé vlastnosti modelů mezi sebou mapovat.

K mapování mezi objekty je doinstalován NuGet balíček „AutoMapper“ ve verzi 9.0.0. Jeho konfigurace je provedena ve třídě `AutoMapping` v projektu `Mapper`. Kromě popsaného nastavení je v projektu přítomná služba `AutoMappingService` implementující metody, jež zajišťují proces mapování z databázových objektů na objekty pro přenos dat a opačně. Zmíněné metody jsou dostupné nad instancí této třídy, která je vytvořena v každém kontroleru užitím konceptu vkládání závislostí (anglicky „dependency injection“).

Samotnou logiku webové aplikace tvoří metody tříd, jež jsou situovány v projektu `Services`. Tyto třídy, nazývané jako „služby“, existují pro každou tabulku databáze neboť obsahují mimo jiné i zdrojový kód pro přístup k datům konkrétní tabulky. Právě za účelem sjednotit opakující se kód v podobě operací pro čtení, zápis, mazání a aktualizaci dat vznikla abstraktní třída `BaseService`. Zde jsou implementovány všechny základní operace nad databází používané službami (`GetById()`, `Create()`, `Delete()`, ...). Definované metody pracují s generickým datovým typem, což je činí znovupoužitelnými.

Důležitou metodou je `ResolveService()`, pomocí které je možné přistupovat k metodám napříč službami. Ku příkladu v metodě třídy `PersonService` lze k metodám třídy `RelationshipService` přistupovat vytvořením proměnné:

```
var relationshipService = ResolveService<IRelationshipService>();
```

Skrze tuto proměnnou je možné přistupovat ke všem metodám, které rozhraní `IRelationshipService` implementuje.

Třídy služeb jsou logicky seskupeny do dvou sekcí, přičemž služby, které manipulují přímo s daty databázových tabulek (entitních množin na obrázku 4.1), jsou uloženy v kořenovém adresáři projektu. Tyto třídy přímo dědí z abstraktní třídy `BaseService`.

Druhou „sekcí“ jsou třídy služeb, které provádějí operace nad tabulkami reprezentujícími vztahy N:N tak, jak jsou popsány v kapitole 5.1. Tyto služby jsou uloženy v adresáři „RelationshipServices“ stejného projektu. Třída `BaseService` předpokládá přítomnost rozhraní `IEntity`, tedy jako premisu staví přítomnost identifikátoru v modelu. Modely reprezentující vztahy však vlastní identifikátor nemají, vznikla tedy jako alternativa třída `BaseRelationshipService`, která svým odvozeným třídám zpřístupňuje modifikované obdobné základní operace.

K obsahu databázových tabulek je ve službách přistupováno pomocí jazyka LINQ, což je jazyk vyvinutý pro dotazování nad daty integrovaný v platformě .NET. Všechny metody, kterým kontrolery předávají řízení, jsou implementovány jako asynchronní užitím klíčových slov `async` a `await`.

Se záměrem maximálně přizpůsobit vyvíjenou aplikaci jejímu účelu byly vytvořeny vlastní výjimky. Tyto výjimky oznamují nesprávný formát objektu nebo neúspěch při jeho hledání v databázi. Dále byla naprogramována mezivrstva (anglicky „middleware“), která všechny vyvolané výjimky zpracuje do formátu JSON a odešle prezentační vrstvě. Tato vrstva je implementována třídou `ExceptionHandling`.

Hlavní problematikou systému jsou sdílené zdroje mezi uživateli. Na rozdíl od běžných informačních systémů, kde má ve většině případů uživatel svá unikátní data, v této práci mezi sebou uživatelé nevědomě data sdílí. Mohou mezi sebou vzájemně sdílet osoby v rodokmenech, vztahy mezi nimi, vzniklé konflikty a tak dále. Jediné, co je pro každého uživatele jedinečné a nemůže být sdíleno, jsou soukromé osobní informace a záznamy o vytvořených rodokmenech (název a typ rodokmenu). Výjimku tvoří privátní rodokmeny (2.2), jejichž obsah náleží pouze jejich autorům. V následujících sekcích je vysvětleno chování klíčových funkcí systému v situacích, kdy je se sdílenými zdroji manipulováno.

Přidávání osoby do rodokmenu

Hlavní operací při manipulaci s rodokmenem je jeho rozšiřování o nové osoby. Z pohledu uživatelského rozhraní autor rodokmenu při přidávání nové osoby nevidí rozdíl, je však potřeba rozlišovat, zda uživatel přidává zcela novou osobu nebo vybral z již existujících záznamů.

Založení nové osoby obstarává metoda `CreatePersonAsync()`. Každá nově vytvářená osoba musí být účastníkem minimálně jednoho vztahu. Pro aplikaci nemá smysl vytvářet osobu, která nemá žádné vztahy, neboť ji nemáme jak zařadit do kontextu rodokmenu. Výjimku tvoří první osoba při zakládání rodokmenu. V této metodě jsou z objektu reprezentujícího osobu vztahy vyjmuty a je nejprve vytvořena samotná osoba s referencí na rodokmen. Vztahy jsou následně vytvářeny iterativně po jednom, neboť každý vztah musí projít procesem popsaném v další podkapitole.

Za předpokladu, že uživatel zvolil již existující osobu, není potřeba ji znovu vytvářet. Zvolená osoba je aktualizována metodou `UpdatePersonAsync()`. Z objektu osoby jsou, podobně jako při vytváření, extrahovány nově přidané vztahy, které lze snadno selektovat díky absenci hodnoty identifikátoru. Tyto vztahy jsou opět jednotlivě vytvářeny postupem popsaném níže. Následně je osoba „přidána“ do rodokmenu vytvořením reference `FamilyTreePerson`.

Vytváření vztahů mezi osobami

Mozkem operací při vytváření vztahů v rodokmenu je ve třídě `RelationshipService` metoda `CreateNewRelationshipInFamilyTreeAsync()`. Metoda vychází z předpokladu, že na vstupu je vztah, který již může v databázi existovat. Vztahem se v takovém případě rozumí záznamy v tabulce `Relationships`, nikoliv vzájemné reference mezi tabulkami.

Tato metoda si klade za cíl řídit proces vytváření nových vztahů v rodokmenu. Před samotným zpracováním plánu pro vytvoření vztahu je potřeba platnost vztahu validovat. Toho je docíleno voláním privátní metody `GeneralRelationshipValidation()`, jenž při neplatnosti vztahu způsobí vyvolání výjimky.

Chování metody se liší na základě typu předaného argumentu. Pokud uživatel přidává vztah *předek–potomek*, existujícím vztahem se rozumí identický záznam v tabulce `Relationships`. Je-li vztah typu *manžel–manželka*, shodný záznam je takový, který oproti předchozí situaci navíc obsahuje i stejné záznamy o svatbě.

Jestliže byl záznam nalezen, další postup už je sjednocen. Pro vztah je zahájeno jeho přidání do příslušného rodokmenu. Kromě reference mezi vztahem a rodokmenem je potřeba založit i nové reference pro všechny konflikty, ve kterých se vztah vyskytuje. Tyto kolize jsou označeny jako nevyřešené, neboť jsou pro daný rodokmen nové. Funkcionalitu zmíněnou v tomto odstavci realizuje metoda `AddExistingRelationshipToFamilyTreeAsync()`.

Pokud žádný vztah odpovídající kritériím nalezen nebyl, bude vztah vytvořen. Pro každý nově přidaný záznam o vztahu je zavolána metoda detekující konflikty. Její implementace je popsána níže.

Detekce kolizí

V kapitole 2.2 jsou popsány dva typy konfliktu, které mohou mezi jednotlivými vztahy nastat. Každý nový vztah je vzápětí ihned po vytvoření podroben kontrole, zda není předmětem konfliktu. Výjimku tvoří pouze vztahy v privátních rodokmenech nebo situace, kde jeden z účastníků vztahu je nedefinovanou osobou. Tyto vztahy na kolize testovány nejsou,

neboť privátní rodokmeny jsou od porovnávání odstíněny (2.2) a nedefinované osoby mohou existovat vždy v jediném rodokmenu. Odhalení obou popsaných typů kolizí provádí metoda `CollisionDetectionAsync()` ve třídě `CollisionService`. Tato metoda je sekvenčně rozdělena na dvě fáze testování.

V první fázi je vykonáváno testování na konflikty typu *kolize generací*. Pro vztah, který je typu *potomek–předek*, probíhá vyhledávání jeho alternativní verze, kde jsou osoby v manželském vztahu a opačně. Jsou-li takové záznamy nalezeny, vztahy jsou v konfliktu. Kolize tohoto typu může obsahovat maximálně tři typy vztahů:

- osoba X je v manželství s osobou Y
- osoba X je předkem osoby Y
- osoba Y je předkem osoby X

Pokud v databázi existuje záznam pro přesně dva z uvedených tří příkladů, musí pro tyto dva vztahy být zaznamenána informace o vzniklé kolizi. V takovémto případě je při testování nutné kolizi vytvořit a zahrnout do ní vztahy, které její vznik způsobují. Typ kolize je nastaven na hodnotu `marriageOrAncestor`. Pro všechny rodokmeny, kde se vztahy vyskytují, je založena reference, která současně nese údaje o jejím vyřešení. Implicitně je vzniklá kolize pro všechny uživatele označena jako nevyřešená nastavením atributu `isSolved` na hodnotu `false`.

Jestliže testovaný vztah je v uvedeném konfliktu a kolize typu `marriageOrAncestor` pro konfliktní vztahy již existuje, je vztah přidán jako předmět kolize. Dále je potřeba vytvořit propojení se všemi rodokmeny stejným způsobem jako v předchozím scénáři.

Druhá fáze detekování konfliktů probíhá pouze pro vztahy typu *předek–potomek*. Napříč rodokmeny je prohledáváno, zda pro stejnou osobu v roli potomka není zadán odlišný předek. Hledány jsou konflikty označené jako *různý předek* (2.2) a jsou rozlišovány podle pohlaví předka. Kolize typu `differentMother` mohou vznikat v situaci, kdy pro jednu osobu existuje ve více rodokmenech odlišná matka. V případě, že je předkem muž, vznikají kolize typu `differentFather` a je tedy v databázi pro určitou osobu uloženo více různých otců.

Pro testovaný vztah jsou vyhledávány vztahy stejného typu s rozdílným identifikátorem předka. V závislosti na počtu výsledků jsou vykonávány rozdílné operace. Jsou-li nalezeni dva různí předci včetně předka testovaného vztahu, je založena nová kolize. Pokud je výsledků více, jsou pouze přidávány reference. Oba dva postupy jsou obdobné jako u řešení kolizí předchozího typu konfliktů.

Zánik kolizí

Pro rodokmeny jsou uchovávány informace o nově vzniklých i již vyřešených kolizích. Existují pouze dva možné scénáře, kdy bude záznam o konfliktu odstraněn z databáze.

Prvním případem je úplné odstranění osoby, jež je předmětem konfliktu. Pokud byla tato osoba předmětem konfliktu a v kolizi už zbývá jediná osoba, informace o kolizi zaniká.

Druhou situací jsou shodná řešení. Vyřeší-li všichni uživatelé rozpor stejně, záznam o kolizi zaniká. K odstranění konfliktu může dojít i ve chvíli, kdy se všichni zainteresovaní uživatelé nevyjádřili. Podmínkou je, že v kolizi musí zůstat jediná osoba. Pro tyto uživatele tak byla výsledná osoba předmětem konfliktu v jejich rodokmenu. Kontrolu na případné odstranění konfliktů v řeší metoda `CheckForCollisionDeleteAsync()` a je volána vždy při odstraňování záznamu z tabulky `Relationships`.

Nalezení podobných osob

Vytvořená aplikace umožňuje uživateli si do vlastního rodokmenu přidávat osoby, které již do systému zadal jiný uživatel. V momentě, kdy vyplní potřebné údaje, je spuštěno vyhledávání alternativních osob. Již existující osoby, které potenciálně vyhovují zadaným požadavkům vyhledává metoda `GetSimiliarPersonsAsync()`. Vyhledáváno je podle následujících kritérií:

- Pohlaví
- Křestní jméno
- Příjmení
- Místo narození (obec)
- Datum narození
- Datum křtu
- Datum úmrtí

Povinnými parametry jsou vlastnosti uvedené v prvním sloupci, kde křestní jméno společně s příjmením jsou vždy v jediné instanci. Při prohledávání databáze jsou ale pro osobu procházena všechna jména i příjmení. V případě tedy, že by například žena měla více příjmení (která nabyla při sňatcích), stačí shoda v jediném, aby parametru příjmení vyhověla.

Data jsou porovnávána pouze v případě, pokud jsou zadána u obou osob. Stejně tak i místo narození jedince. Při srovnávání dat je nastavena roční tolerance a za validní jsou tedy považována všechna data, která spadají do časového okruhu jednoho roku. Porovnání s roční tolerancí zajišťuje metoda `AreDatesSubtractLessThanYear()`, která na vstupu očekává dvě kalendářní data. Není-li datum u jedné z osob (či obou) zadáno, porovnání je zhodnoceno jako úspěšné.

K tomu, aby mohla být osoba označena jako podobná a nabídnuta uživateli, je tedy zapotřebí:

- shoda v pohlaví
- křestní jméno musí prvkem množiny jmen existující osoby
- příjmení musí prvkem množiny příjmení existující osoby
- shoda v zadaných datech s příslušnou tolerancí

Uživateli nejsou nabízeny osoby, které už v rodokmenu jsou, a to ani za předpokladu, že by vyhověly všem požadavkům. Osoby označené jako privátní jsou z vyhledávání vyloučeny a nemůžou tak být nabídnuty jinému uživateli.

Veškeré metody zajišťující nalezení podobných osob jsou součástí služby `PersonService`.

Nahrazování osob rodokmenu

Realizovat nahrazení osoby, která se může vyskytovat i uprostřed rodokmenu, může být užitečné v následujících dvou situacích:

1. uživatel má ve svém rodokmenu kolizi typu *rozdílný předek*, kterou se chystá vyřešit. Nazná, že současná osoba, jež je součástí konfliktu, není správným řešením a vybere z ostatních osob kolize, které jsou mu aplikací nabídnuty. V takovém případě je potřeba původní osobu v rodokmenu nahradit a zajistit zachování všech vztahů.

2. uživatel chce nedefinovanou osobu v rodokmenu doplnit o informace a nahradit ji za plnohodnotnou osobu s vyžadovanými parametry.

Druhý uvedený scénář lze ještě dále rozvětvit v závislosti na tom, zda je nedefinovaná osoba vystřídána osobou již existující vybranou při vyhledávání podobných osob, nebo zda uživatel definoval osobu novou.

Uvažujme v tuto chvíli, že si uživatel nevybral žádnou z již existujících osob a chce založit osobu novou. Protože nedefinovaná osoba může existovat vždy pouze v jediném rodokmenu a je již součástí všech potřebných vztahů, je možné přistoupit k řešení zachováním této osoby. Pouze je pro osobu vytvořeno křestní jméno, příjmení a jsou nahrazena všechna data, jsou-li uživatelem zadána. Hodnota příznaku `isPrivate` je určena typem rodokmenu a příznak `isUndefined` je změněn na hodnotu `false`. Substituce toho typu je řešena metodou `ReplaceUndefinedPersonInFamilyTreeAsync()`.

Řeší-li uživatel kolizi nebo nahrazuje nedefinovanou osobu již existující osobou, je chování aplikace stejné. V obou případech je žádoucí původní osobu odebrat a do zachovaných vztahů zasadit osobu novou. Vztahy osoby však zůstanou udrženy pouze z pohledu uživatele, neboť nově zobrazená osoba má odlišný identifikátor a je potřeba tuto problematiku řešit. Proces nahrazení osoby zajišťuje metoda `ReplacePersonInFamilyTreeAsync()`. Pro původní osobu jsou získány všechny vztahy, kterých se v upravovaném rodokmenu účastnila. Zmíněné vztahy jsou duplikovány a identifikátor bývalé osoby je v nich nahrazen primárním klíčem osoby nové. Každý vztah je následně předán k vytvoření (proces vytváření vztahů mezi osobami je popsán v oddělené sekci této kapitoly).

Původní osoba je odebrána z upravovaného rodokmenu metodou, která zajistí odstranění všech referencí, včetně případných participací v kolizích. Následně je vytvořeno spojení rodokmenu s novou osobou a proces výměny osob je tak dokončen.

Výsledkem je nahrazení jedné konkrétní osoby v rodokmenu, navázání nové do všech potřebných vztahů a smazání z rodokmenu všech konfliktů, kterých byla původní osoba součástí.

Konkatenace rodokmenů

Při nahlížení uživatele do cizích rodokmenů má možnost si jejich obsah zkopírovat a přidat do vlastního rodokmenu, který vytváří. Spojení rodokmenů zajišťuje asynchronní metoda `ConcatenateFamilyTreesAsync()` ve třídě `FamilyTreeService`. Aby ke konkatenaci dvou rodokmenů mohlo dojít, musí nejprve být splněny následující podmínky:

- Zdrojový a cílový rodokmen, do kterého kopírujeme, se musí prolínat alespoň v jedné osobě. V případě, že by společná osoba absentovala, není možné rodokmeny propojit.
- Zdrojový rodokmen musí mít veřejný typ, cílový rodokmen nesmí být privátní, může tedy být nastaven pouze jako veřejný nebo neveřejný (více k typům rodokmenů v kapitole 2.2).
- Vztahy v rodokmenech nesmí být mezi sebou vzájemně v kolizi. Jedinou výjimkou je případ, kdy příčinou kolize je v obou rodokmenech stejný vztah (jsou tedy společně v konfliktu ještě s dalším rodokmenem).

Validaci těchto parametrů provádí metoda `CanConcatenateAsync()`. Po provedeném ověření jsou získány všechny vztahy pro současný stav zdrojového i cílového rodokmenu. Kopírování elementů do rodokmenu je pomyslně rozděleno na tři části.

První část iteruje skrze všechny vztahy ve zdrojovém rodokmenu a vytváří na ně reference ve zvoleném rodokmenu přihlášeného uživatele. Některé vztahy už v cílovém rodokmenu mohou být přítomny, ty jsou v takovém případě přeskočeny. Komplikace přicházejí s nedefinovanými osobami, které mohou existovat pouze v jediném rodokmenu a nesmí tak být na ně přidána reference do jiného rodokmenu. Pokud je účastníkem vztahu nedefinovaná osoba, je v cílovém rodokmenu hledána její alternativa. Existuje-li nedefinovaná osoba se stejnými vztahy, bude původní osoba namapována na alternativní již existující osobu. Toho je docíleno užitím kolekce slovníku, kdy identifikátor původní osoby vystupuje jako klíč a identifikátor nedefinované osoby v cílovém rodokmenu jako hodnota. Pokud osoba není nalezena, je vztah uložen do kolekce a řešen později.

Ve druhé fázi jsou iterativně vytvořeny reference na kolize pro všechny nové vztahy, které do rodokmenu uživatele přibyly. Tyto reference představuje tabulka `FamilyTreeCollision`. Bez ohledu na fakt, zda majitel zdrojového rodokmenu dané kolize už vyřešil či nikoliv, pro uživatele rozšiřovaného rodokmenu jsou všechny nově nabyté kolize označeny jako nevyřešené.

Ve třetí části jsou iterativně procházeny všechny osoby zdrojového rodokmenu. Jestliže už v rodokmenu před konkatenací osoba existovala, iterační krok je přeskočen, v opačném případě je vytvořena nová reference vztahu `FamilyTreePerson`. Scénář se opět liší, pokud osoba není definována. Jestliže se tuto osobu podařilo v první části namapovat na již existující osobu, jsou pro ni pouze vytvořeny nové vztahy. Pokud namapování nebylo úspěšné, je vytvořena nová nedefinovaná osoba, které jsou přiřazeny zduplikované vztahy zdrojové osoby.

Vzniklý rodokmen je vyhodnocen funkcí `ReplaceUndefinedPersonByNewlyDefined()`. Jak už název funkce napovídá, jejím primárním cílem je nalézt osoby, které byly v původním rodokmenu nedefinovány, ve zdrojovém rodokmenu však identita těchto osob byla známa. Pokud tedy vztahy nedefinované osoby byly podmnožinou vztahů osoby se známou identitou, je touto osobou v rodokmenu nahrazena.

Na závěr je kontrolována platnost vlastnosti `StartPersonId` rodokmenu. Tento identifikátor definuje střena, od kterého je rodokmen vykreslován (více v kapitole 5.3). Pokud tato osoba má nově předky, tato vlastnost je přepsána hodnotou rodokmenu, ze kterého byl obsah přejat. Zmíněnou kontrolu a případnou změnu počáteční osoby rodokmenu implementuje metoda `ChangeStartPersonAfterConcatenate()`.

Po dokončení konkatenace je provedena kontrola, zda všechny zkopírované osoby patří do rodu cílového rodokmenu. Osoby, které nejsou součástí rodu, jsou vzápětí z rodokmenu odstraněny.

Ukládání hesel

Ukládání hesel v otevřené podobě je v moderních aplikacích nepřípustné a nejčastějším řešením je jejich uložení v šifrované podobě. Heslo je v takovém případě předáno určitému kryptografickému algoritmu a jeho čitelný obsah je zaměněn za řetězec bytů, které dávají smysl pouze v kontextu tzv. kryptografické hašovací funkce. Tato funkce algoritmicky vypočítává a vrací vygenerovaný sled bytů, z něhož v závěru není možné zpět vykonstruovat původní podobu hesla.

V této práci je vytváření šifry hesla docíleno využitím kryptografické funkce PBKDF2, která pro generování klíče využívá iterativní volání pseudonáhodného generátoru čísel a takzvanou kryptografickou sůl. Tato „sůl“ je pouze polem náhodně vygenerovaných bitů, které má za účel zmenšit pravděpodobnost identického hashe pro stejný vstup (v tomto případě

uživatelské heslo). Celý tento proces v aplikaci zajišťuje třída `PasswordService`. Minimální doporučený počet iterací kryptografické funkce je 1000, s větším počtem iterací roste bezpečnost uloženého klíče, současně však vznikají vyšší nároky na výpočetní sílu hardware.

V implementaci je zvolena hranice 10000 iterací, kryptografický hash představuje pole o velikosti 32 bytů, kterému sekunduje kryptografická sůl o stejných parametrech. Pro porovnání uloženého hesla s uživatelským vstupem je nutné v databázi uchovávat informaci o kryptografickém hashi i použité „soli“, uloženy jsou však ve formátu řetězce znaků Base64 a před použitím v aplikaci jsou vždy z tohoto formátu dekódovány.

Proces autentizace a autorizace

Implementovaná webová aplikace pracuje s uživatelskými daty, jejichž vlastníkem je daný přihlášený uživatel a přístup k nim musí být regulovaný. Vznikl tedy požadavek na ověření uživatele v podobě autentizace a autorizace.

Při uživatelově přihlášení nebo registraci získá aplikace prezentační vrstvy informace o přiřazeném identifikátoru a zašifrované heslo ve formátu řetězce znaků. Přijaté údaje ihned zpracuje a uloží do lokální paměti prohlížeče klienta ve formátu:

```
auth: {id: 1, hash: "HD8qBSyY9d2eDzuTbCnWhWX4sVuL57hhmP08U8deUXs="}
```

Tyto unikátní údaje jsou poté připojeny do HTTP hlavičky každého dotazu vyslaného na server a uživatel se jimi tak prokazuje při autentizaci. Kryptografická sůl však není součástí žádného z dotazů, je bezpečně uložena v databázi a načítána pouze při procesu ověřování uživatele.

Na straně webového API existuje mezivrstva aplikace (anglicky „middleware“), která zachytává příchozí požadavky ještě před vyvoláním příslušné obsluhy (předání řízení konkrétnímu kontroleru). Z hlavičky požadavku jsou extrahovány identifikační údaje a kryptografický hash je porovnán se záznamem osoby z databáze. Pokud údaje souhlasí, je porovnáváno, zda daná osoba je oprávněna vykonat danou operaci. V případě že ano, je nastaven autentizační tiket a osoba je považována za ověřenou.

I v případě, že ověření identity uživatele selže, řízení je dále předáno kontroleru, kterému byl požadavek adresován. Vyžadují-li kontrolery validaci uživatele, jsou označeny atributem `[Authorize]`. Pokud je tímto atributem označena třída, všechny kontrolery, které třída implementuje, vlastnost dědí. V této třídě se mohou vyskytovat kontrolery označené atributem `[Allow Anonymous]`, které zděděnou vlastnost anulují.

Pokud kontroler ověření uživatele nevyžaduje, bude spuštěn i v případě selhání autentizace, v opačném případě je vrácen autentizační chybový kód 401.

5.3 Prezentační vrstva

Prezentační vrstva aplikace zajišťuje zprostředkování výsledného vzhledu webové stránky a zpracovávání uživatelovy interakce s webem. Této úlohy se v implementaci ujímá aplikace vytvořená ve webovém rámci Angular (3.2.4).

Vytvořený zdrojový kód v tomto projektu lze rozdělit do tří hlavních částí:

- Komponenty (umístěné ve složce „components“)
- Služby (umístěné ve složce „services“)
- Modely (umístěné ve složce „models“)

Jednotlivé části jsou vysvětleny níže, přičemž jsou popisovány v pořadí podle rozsáhlosti.

Modely jsou definicí struktury objektů, se kterými je napříč aplikací pracováno. Aplikace od serveru přijímá odpovědi, které jsou nejčastěji DTO objekty ve formátu JSON. Aby s daty bylo možné v prezentační vrstvě pracovat, je potřeba DTO objekty a jejich vlastnosti namapovat na vlastnosti, které aplikace zná. Pro každý takový objekt tedy v prezentační vrstvě existuje ekvivalentní model.

Kromě modelů, které jsou potřeba pro přenos dat mezi prohlížečem uživatele a serverem, jsou v aplikaci definovány modely nezbytné pro vykreslování rodokmenů.

Důležitou součástí implementace jsou třídy nazývané jako služby. Jejich nejčastější úlohou je zprostředkování dat mezi klientskou aplikací a serverem. Pro každou službu, která komunikuje se serverem, existuje na straně serveru vždy jeden adekvátní kontroler. Oba zpravidla manipulují se stejnými modely a zasílají si data pouze mezi sebou. Počet služeb komunikujících po síti tak odpovídá počtu kontrolerů.

Všechny tyto služby využívají pro zasílání požadavků metody implementované ve třídě `QueryService`. Tato třída zprostředkovává metody pro zasílání GET, POST, PUT a DELETE požadavků, přičemž ke každé žádosti připojuje HTTP hlavičku. Důležitým obsahem hlavičky jsou autentizační údaje, kterými uživatel na serveru prokazuje svoji identitu. Hash a identifikátor je z paměti prohlížeče získán díky metodě `getHash`, kterou implementuje služba `AuthorizationService`.

Nejdůležitější složkou aplikace jsou komponenty, protože právě ony formulují výsledné zobrazení webové stránky pro uživatele. V této práci jsou zdrojové kódy komponent rozděleny do třech různých složek v závislosti na účelu, který komponenta plní.

Účely se rozumí:

1. *práce s daty* – hlavní komponenty, které vytvářejí dynamický obsah webových stránek. Kontrolují a zpracovávají uživatelské vstupy, zajišťují personalizované vykreslování. Tyto komponenty manipulují s daty přijatými ze serveru, zprostředkovávají je uživateli a využívají tedy služby.

2. *směrování* – každá taková komponenta má přiřazenou svou vlastní URL adresu, která slouží pro směrování v rámci aplikace. Samy o sobě nemají žádný vlastní obsah, pouze vnořují další komponenty, které vytvářejí výsledný vzhled stránky.

3. *zobrazení* – komponenty, které neobsahují žádnou nebo pouze minimální logiku, mají pouze statický obsah.

Hlavní části uživatelského rozhraní aplikace tvoří komponenty `UserDashboardComponent` a `FamilytreeComponent`. Tyto dvě komponenty jsou současně kontejnery pro další vnořenou funkcionalitu a spravují úseky aplikace, se kterými bude uživatel pracovat nejvíce.

První zmíněná komponenta má na starosti správné zobrazení profilu přihlášeného uživatele. Komponenta `FamilytreeComponent` zaštiťuje z hlediska uživatelského rozhraní vykreslení rodokmenu a veškerou manipulaci s ním.

Pokud dojde v aplikaci k chybě nebo operace na serveru vrátí výjimku, je uživatel o této skutečnosti informován přesměrováním na chybovou stránku komponenty `ErrorComponent`. To neplatí pro případ, kdy uživatel chce provést akci, na kterou nemá dostatečná oprávnění, tehdy je odkázán na specifickou komponentu `ErrorAuthorizationComponent`. Chybová

hlášení zachytává a zpracovává třída `GlobalErrorHandler`, která vzniklou chybu vypíše do vývojářské konzole a vzápětí uživatele přesměruje na příslušnou chybovou stránku.

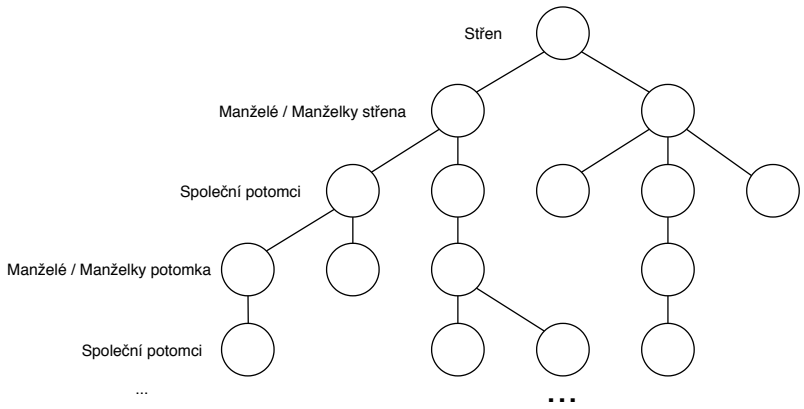
Níže jsou popsány hlavní myšlenky prezentační vrstvy, které vytvářená práce implementuje. Tyto principy provádí výhradně komponenty, které pracují s daty.

Vykreslování rodokmenů

Vykreslování rodokmenů je v grafickém uživatelském rozhraní tím nejdůležitějším procesem aplikace. Na úvod je vhodné zmínit, že rodokmen je konstruován postupným procházením jednotlivých vztahů počínaje vztahy aktuálního střena. K samotnému vykreslení je užita knihovna `dTree` (3.2.5), které jsou dodána data v definovaném formátu.

V prvním kroku jsou ze serveru získány informace o všech vztazích, které se v rodokmenu nacházejí, včetně osob, které se jich účastní. Vzhledem ke kruhové závislosti je možné pro kteroukoliv osobu získat všechny její vztahy pro aktuální rodokmen. Toho je využito v následujícím kroku.

Ve chvíli, kdy má aplikace dostupná potřebná data, je potřeba vykonat jejich zpracování do příslušného formátu. Tímto formátem se rozumí hierarchická struktura stromu, která je schématicky vyjádřena na obrázku 5.1.



Obrázek 5.1: Formát dat pro vykreslování rodokmenu

Kořenem stromu je střen, jehož bezprostředními následníky jsou manželé/manželky. Jejich následníky jsou zase společní potomci. Tito potomci mají své manželské svazky a děti. Postup se dále opakuje až ke kořenovým uzlům stromu.

Této struktury je docíleno rekurzivním voláním metody `createDataTreeContent()`, která je sestavením pověřena. Metoda na vstupu očekává osobu (pro první volání střena), ke které jsou najiti všichni manželé/manželky. V závislosti na pohlaví jedince je osobě přiřazena třída s formátováním, která určuje způsob jejího vykreslení v rodokmenu. Pro osobu na vstupu a konkrétní choť jsou nalezeni všichni společní potomci. Následně pro každého potomka metoda rekurzivně volá sebe sama a postup opakuje.

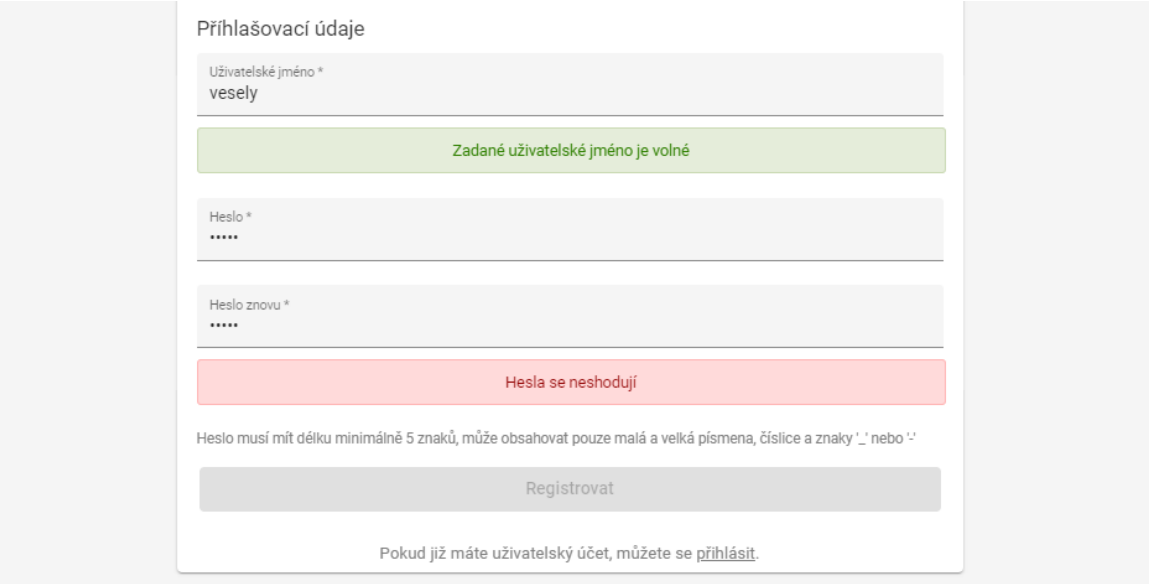
Všechny úrovně rodokmenu a osoby v nich jsou postupně od koncových uzlů vnořovány do objektu předchůdce. Výsledkem je tak jediný objekt reprezentující zakladatele rodu. Za účelem formátování dat pro vykreslení byly vytvořeny modely `RenderPerson` a `RenderPersonMarriage`, které na sebe vzájemně odkazují a reflektují strukturu stromu podle dokumentace knihovny.

Na závěr jsou data předána metodě `renderFamilyTree()`, která zajišťuje vykreslení rodokmenu a jeho zobrazení uživateli. Tato metoda spolupracuje s již zmiňovanou knihovnou `dTree` a zachytává zpětná volání knihovny v případech, kdy uživatel klikl na některou z osob.

Kontroly uživatelských vstupů

Napříč programem je dohlíženo na konání uživatele s cílem zamezit nesprávnému nakládání s aplikací. Jedním ze způsobů, jak jsou jeho akce kontrolovány v prezentační vrstvě, je testování validity zadávaných dat ve formulářích. Platnost těchto uživatelských vstupů je prověřována užitím regulárních výrazů a uživateli je potvrzovací tlačítko zpřístupněno až ve chvíli, kdy všechny vstupy vyhovují parametrům (příkladem může být obrázek 5.2).

Větší míru validace vyžaduje formulář při registraci nového uživatele. Mimo kontrolu regulárními výrazy je ověřováno, zda je přihlašovací jméno v rámci databáze unikátní a zda se zadaná hesla shodují.



Obrázek 5.2: Screenshot aplikace – Část formuláře pro registraci

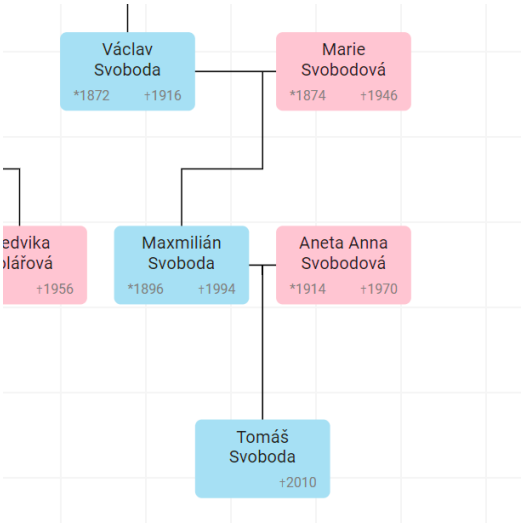
Na obrázku 5.2 je k vidění část registračního formuláře, který je nevyhovující a uživateli je tak zamezeno pokračovat v registraci.

S každým nově přidaným či odebraným znakem uživatelského jména je asynchronně zasílán požadavek na server, kde příslušná metoda otestuje jeho jedinečnost v rámci databáze. Ihned při vyplňování tak uživatel ví, zda je jím zvolené přihlašovací jméno volné.

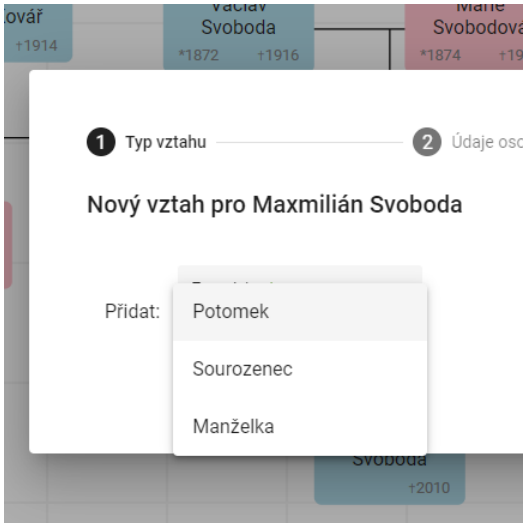
S účelem zabránit nechtěným přehmatům při zadávání hesla je uživatel vyzván k jeho opakovanému vyplnění. Zadané vstupy se musí shodovat, jinak není uživateli dovoleno pokračovat.

Celý proces registrace zajišťuje komponenta `UserRegisterComponent`.

Dalším důležitým bodem kontroly správného užívání aplikace je regulace vztahů, ve kterých mohou být nové osoby přidány. Do rodokmenu mohou být přidáni pouze lidé, kteří patří do daného rodu. Těmi například nejsou rodiče přivdané ženy.



Obrázek 5.3: Výstřižek rodokmenu



Obrázek 5.4: Přidávání nové osoby

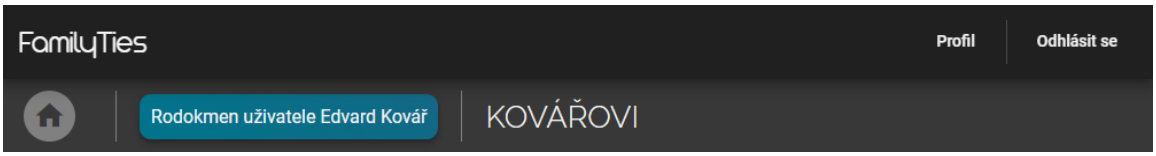
Na obrázku 5.3 je zachycena část rodokmenu, jenž obsahuje osobu „Maxmilián Svoboda“, pro kterou je přidáván nový vztah. Vedle tohoto výstřižku je zobrazeno samotné modální okno pro přidávání vztahu. Na snímku 5.4 lze vidět, že pro osobu už nelze přidat otce a matku, neboť tyto osoby už v rodokmenu přidány jsou. Uživateli jsou tak povoleny pouze vztahy, které mohou být v rodokmenu založeny. Pro osoby přivdané/přiženěné do rodokmenu mohou být přidáni pouze potomci, kteří byli zplozeni s pokrevním členem rodokmenu.

Při spuštění procesu přidávání nové osoby je tedy vždy ověřeno, které vztahy jsou pro rodokmen platné. Průběh přidávání nové osoby zaštiťuje třída `PersonAddComponent`.

Nahlížení do cizích rodokmenů

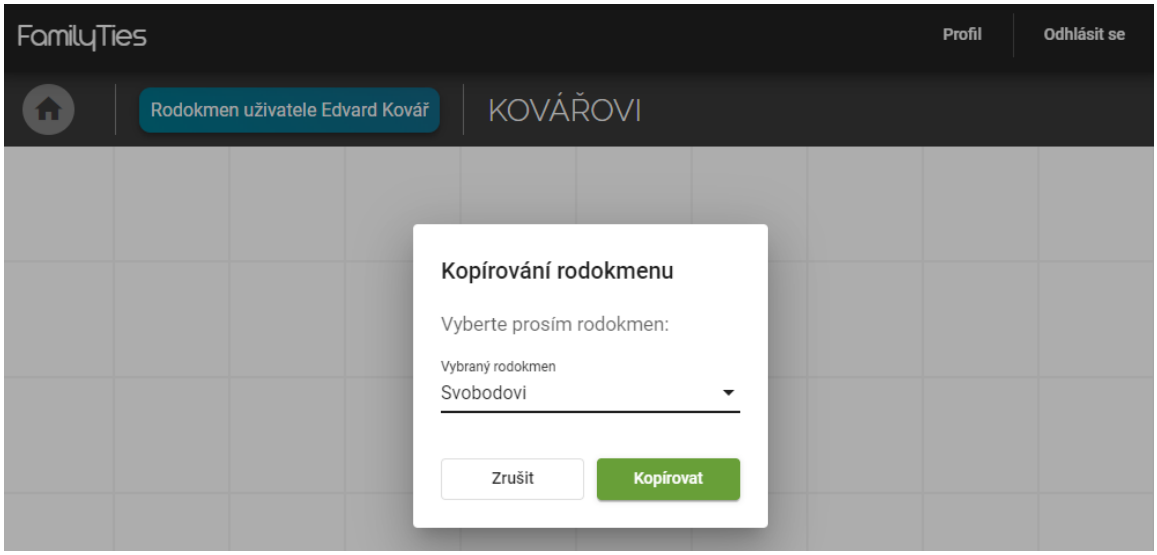
Uživatel má možnost nahlížet do cizích rodokmenů, které jsou nastaveny jako veřejné. S těmito rodokmeny nemůže nijak manipulovat a provádět žádné změny. Pokud by chtěl rodokmen upravit, má možnost jeho obsah nakopírovat a spojit s některým z vlastních rodokmenů.

Jak je možné vidět na obrázku 5.5, ovládací prvky v horním panelu jsou uživateli skryty (lze porovnat s obrázkem 4.3, kde je k vidění zobrazení pro autora rodokmenu). Aplikace navíc na server zašle dotaz na jméno autora, které je vyobrazeno nalevo od názvu rodokmenu.



Obrázek 5.5: Screenshot aplikace – Horní panel při zobrazení cizího rodokmenu

K dispozici uživatel nemá pochopitelně ani panel pro manipulaci s osobami, namísto něj je zobrazeno tlačítko pro kopírování rodokmenu. Pokud na něj uživatel klikne, naskytne se mu pohled zachycený na obrázku 5.6.



Obrázek 5.6: Screenshot aplikace – Modální okno pro kopírování rodokmenu

Na server je zaslán požadavek, který jako odpověď očekává rodokmeny uživatele vyhovující požadavkům pro konkatenaci (popsány v kapitole 5.2 v sekci „Konkatenace rodokmenů“). Prezentační vrstva poté uživateli poskytne výběr z těchto rodokmenů. V případě, že žádný rodokmen, do kterého je možné kopírovat, ve své sbírce nemá, je uživateli zobrazena informace o nemožnosti kopírování včetně důvodů, proč tomu tak je. Pokud uživatel provede nakopírování obsahu rodokmenu, je po dokončení aplikací automaticky přesměrován na cílový rodokmen.

5.4 Testování

Systém, jenž je předmětem této práce, byl po celý čas vývoje testován. S každou přidanou dílčí částí byla podrobně testována její funkčnost, v delších časových intervalech činnost celého programu. Popsaná kontrola správného chování systému byla prováděna programátorem.

V samotném závěru vývoje, kdy systém splňoval všechny požadované body, bylo provedeno uživatelské testování. Systém tak byl podroben testování nezávislými uživateli, jenž byli předem seznámeni se záměrem implementované aplikace a jejími funkcemi.

Správné chování výsledné aplikace bylo kontrolováno na vytvořených demonstračních datech. Pro účely testování byly založeny dva uživatelské účty a v souhrnu čtyři rodokmeny, dva na každém účtu. Pokryty byly všechny možnosti nastavení soukromí rodokmenů, jež jsou na výběr (popsané v kapitole 2.2), přičemž oba uživatelé disponovali veřejným rodokmenem. Dohromady bylo do systému vloženo bezmála sto dvacet různých osob. Založené osoby měly proměnlivý počet křestních jmen a příjmení. Bylo tak testováno i přidávání/odebírání jmen osob a vyhledávání v aplikaci DEMoS při každé jejich aktualizaci. Některé z vytvořených osob se vyskytovaly napříč více rodokmeny. Díky tomu docházelo k prolínání rodokmenů a vznikla tak možnost testovat korektní vznik konfliktů a kopírování rodokmenů mezi uživateli.

Použitá testovací data byla z databáze exportována a jsou uložena na přiloženém paměťovém médiu.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vyvinout webovou aplikaci včetně grafického uživatelského rozhraní, která by poskytovala možnost jednotlivým uživatelům spravovat své vlastní rodokmeny. Systém byl vytvořen v rámci souběžně vyvíjeného genealogického projektu DEMoS na Fakultě informačních technologií VUT v Brně.

K dosažení výsledné aplikace bylo nejprve potřeba se seznámit s problematikou vedení matričních záznamů a důkladně navrhnout vyhovující databázové schéma. Před započetím implementace bylo nezbytné nastudovat principy tvorby webových aplikací a zkoumat vhodné technologie. Pro realizaci implementační stránky práce jsem se musel seznámit s platformou .NET a programovacím jazykem C#. K vytvoření uživatelského rozhraní bylo zapotřebí porozumět konceptu a osvojit si praktiky aplikačního rámce Angular.

Záměr práce se podařilo naplnit a výsledkem je tak funkční systém pro správu rodokmenů. Základem vytvořeného systému je databáze schopná uchovávat informace pro všechna genealogická schémata. Současně uvažuje i neurčitost při navazování na matriční záznamy a nejistotu uživatele při vytváření rodokmenů. Pro aplikaci bylo vyvinuto grafické uživatelské rozhraní, které uživateli poskytuje možnost vytvářet své vlastní rodokmeny s různými možnostmi nastavení soukromí. Systém úspěšně odhaluje a vyhodnocuje konflikty vznikající mezi rodokmeny a nabízí každému uživateli zvolit vlastní řešení. Uživatelské rozhraní dále disponuje vyhledáváním relevantních osob, které je prováděno automaticky při přidávání nových záznamů v rodokmenu. Nad rámec zadání byla po konzultaci s vedoucím práce aplikace doplněna o možnost kopírování veřejných rodokmenů mezi uživateli.

Program popsaný v této technické zprávě byl vytvářen souběžně s odkazovaným projektem s myšlenkou jejich vzájemného propojení. Předmětem následujících měsíců je tak nasazení vytvořeného systému do produkčního prostředí. Dále je do budoucna uvažováno o sjednocení uživatelských účtů napříč mnou vyvinutým programem a aplikací DEMoS. Uživatel by tak reálně měl jediné přihlašovací údaje pro obě aplikace.

V práci by bylo možné pokračovat ku příkladu přidáním některých z mnou navržených rozšíření. Do aplikace by bylo zajímavé přidat možnost exportování a importování rodokmenů z/do formátu GEDCOM, se kterým konkurenční nástroje dokáží pracovat. Pro záznamy osob z přelomu 20. a 21. století a mladšími lze uvažovat o možnosti rozšíření aplikace o přidávání registrovaných partnerství či adoptovaných potomků do rodokmenu. Z technického hlediska by bylo bezpečnější a uživatelsky přívětivější použít pro autentizaci protokol OAuth2. Uživatelé by se tak mohli do aplikace registrovat a přihlašovat pomocí služeb podporujících tento protokol.

Výsledná aplikace bude pracovat především s matričními záznamy z 20. století a staršími. S přibývajícími digitalizovanými záznamy aplikace DEMoS se stane vyvinutý systém

komplexnějším nástrojem, neboť se bude zvyšovat pravděpodobnost navázání osob s konkrétním matričním záznamem.

Literatura

[1] *Acta publica* [online]. Moravský zemský archiv v Brně [cit. 2020-05-13]. Dostupné z: http://actapublica.eu/.

[2] Albahari, J. a Albahari, B. *C# 7.0 in a Nutshell*. 1. vyd. O'Reilly Media, Inc., Říjen 2017. ISBN 978-1-491-98765-0.

[3] Ali, S. M. *Client-Server Model* [online]. GeeksForGeeks, 2019 [cit. 2020-03-27]. Dostupné z: https://www.geeksforgeeks.org/client-server-model/.

[4] Angular. *Introduction to Angular concepts* [online]. Angular [cit. 2020-02-23]. Dostupné z: https://angular.io/guide/architecture.

[5] Bakni, M. *3-tier client/server model architecture* [online]. Wikimedia, Srpen 2017 [cit. 2020-04-01]. Dostupné z: https://commons.wikimedia.org/wiki/File:Client-Server_3-tier_architecture_-_en.png.

[6] Bittner, J. *Úvod do CSS preprocesoru Sass* [online]. ITnetwork.cz, Duben 2014 [cit. 2020-02-26]. Dostupné z: https://www.itnetwork.cz/html-css/webove-portfolio/tutorial-moderni-webove-portfolio-sass.

[7] Borůvka, F. *Genealogie v praxi* [online]. Technická univerzita v Liberci, Centrum dalšího vzdělávání, říjen 2016 [cit. 2020-03-07]. Dostupné z: https://www.cdv.tul.cz/wp-content/uploads/2020/01/U3V_9_genealogie_tabulky.pdf.

[8] Běhálek, M. *Programovací jazyk C#* [online]. Výukový materiál. 2007 [cit. 2020-02-20]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text.pdf.

[9] Chappell, D. *Introducing .NET* [online]. CODE Magazine, 2006 [cit. 2020-02-19]. Dostupné z: https://www.codemag.com/Article/070013/Chapter-1-Introducing-.NET.

[10] De la Torre, C. *What is .NET Core 5 and ASP.NET 5 within .NET 2015 Preview* [online]. Microsoft, Listopad 2014 [cit. 2020-02-23]. Dostupné z: https://devblogs.microsoft.com/cesardelatorre/what-is-net-core-5-and-asp-net-5-within-net-2015-preview/.

[11] Dresler, R. *Vícevrstvé architektury aplikací* [online]. Duben 2011 [cit. 2020-04-01]. Dostupné z: http://www.robertdresler.cz/2011/04/vicevrstve-architektury-aplikaci.html.

[12] *Entity Framework Tutorial* [online]. [cit. 2020-02-23]. Dostupné z: https://www.entityframeworktutorial.net/.

[13] Ferguson, N. *What's The Difference Between Frontend And Backend Web Development?* [online]. 2020 [cit. 2020-02-16]. Dostupné z: https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/.

[14] Groom, A. *Introduction to Single-Page Applications (SPA)* [online]. DZone, Červenec 2018 [cit. 2020-02-25]. Dostupné z: https://dzone.com/articles/what-is-a-single-page-application.

[15] Herceg, T. *Úvod do .NET frameworku* [online]. dotNETportal.cz, Duben 2009 [cit. 2020-02-16]. Dostupné z: https://www.dotnetportal.cz/clanek/125/Uvod-do-NET-Frameworku.

[16] JReport. *3-Tier Architecture: A Complete Overview* [online]. Jinfonet Software, Inc., Prosinec 2017 [cit. 2020-03-27]. Dostupné z: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/.

[17] Kosek, J. *Historie a vývoj HTML* [online]. 2013 [cit. 2020-03-05]. Dostupné z: http://htmlguru.cz/uvod-historie.html.

[18] Kroupová, T. *Kniha: Cesta ke kořenům objevování tajemství tvého rodu.* Olomouc, CZ, 2018. Diplomová práce. Univerzita Palackého v Olomouci, Fakulta pedagogická. Dostupné z: https://theses.cz/id/1iqzbf/Kroupova_Tereza_DP.pdf.

[19] Kucheriavy, A. *Results on Internet (Roi): Secrets of Successful Business Websites.* 1. vyd. AuthorHouse, Říjen 2014. ISBN 978-1-4969-3351-5.

[20] Kumar, A. *.NET Framework vs .NET Core* [online]. DZone, Září 2019 [cit. 2020-04-09]. Dostupné z: https://dzone.com/articles/net-framework-vs-net-core.

[21] Kvapil, J. *Úvod do TypeScriptu* [online]. ITnetwork.cz, 2018 [cit. 2020-02-26]. Dostupné z: https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu.

[22] Lander, R. *Announcing .NET Core 3.1* [online]. Microsoft, Prosinec 2019 [cit. 2020-04-09]. Dostupné z: https://devblogs.microsoft.com/dotnet/announcing-net-core-3-1/.

[23] LearnFrenzy Team. *Top 25+ Angular 2 Interview Questions & Answers* [online]. LearnFrenzy, Srpen 2017 [cit. 2020-02-25]. Dostupné z: https://learnfrenzy.com/blog/top-25-angular-2-interview-questions--answers.

[24] Margaret Rouse and Cameron McKenzie and Andrew Trubac and others. *HTML (Hypertext Markup Language)* [online]. TheServerSide, 2005 [cit. 2020-02-27]. Dostupné z: https://www.theserverside.com/definition/HTML-Hypertext-Markup-Language.

[25] MariaDB. *MariaDB versus MySQL: Compatibility* [online]. MariaDB, leden 2020 [cit. 2020-03-16]. Dostupné z: https://mariadb.com/kb/en/mariadb-vs-mysql-compatibility/.

[26] Marston, T. *What is the 3-Tier Architecture?* [online]. Říjen 2012 [cit. 2020-04-01]. Dostupné z: https://www.tonymarston.net/php-mysql/3-tier-architecture.html.

[27] Massi, B. *Understanding .NET 2015* [online]. Microsoft, Únor 2015 [cit. 2020-02-23]. Dostupné z: https://devblogs.microsoft.com/bethmassi/understanding-net-2015/.

[28] Mikoluk, K. *XAMPP Tutorial: How to Use XAMPP to Run Your Own Web Server* [online]. Udemy, září 2013 [cit. 2020-02-25]. Dostupné z: https://www.udemy.com/blog/xampp-tutorial/.

[29] Máca, J. *Úvod do Angular* [online]. ITnetwork.cz, Leden 2019 [cit. 2020-02-25]. Dostupné z: https://www.itnetwork.cz/javascript/angular/zaklady/uvod-do-angular-frameworku.

[30] Pecháček, J. *Genealogické diagramy aneb co je to vlastně rodokmen? Část II.* [online]. 2015. 2017-01-10 [cit. 2020-03-07]. Dostupné z: https://www.odkudjsme.cz/blog/genealogicke-diagramy-aneb-co-je-to-vlastne-rodokmen/.

[31] Pecháček, J. *Jak vypadají matriční záznamy* [online]. 2015. 2018-07-24 [cit. 2020-03-11]. Dostupné z: https://www.odkudjsme.cz/blog/jak-vypadaji-matricni-zaznamy/.

[32] Peterka, J. *Model klient/server* [online]. eArchiv.cz, 1996 [cit. 2020-03-27]. Dostupné z: http://www.earchiv.cz/a96/a609k150.php3.

[33] Piiroinen, J. *Overview of the Common Language Infrastructure* [online]. Wikipedia, Únor 2008 [cit. 2020-02-20]. Dostupné z: https://en.wikipedia.org/wiki/File:Overview_of_the_Common_Language_Infrastructure.svg.

[34] Roth, D., Anderson, R. a Luttin, S. *Introduction to ASP.NET Core* [online]. Microsoft, Listopad 2019 [cit. 2020-04-09]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1.

[35] Rouse, M. a Fitzgibbons, L. *Front end and back end* [online]. 2006 [cit. 2020-02-16]. Dostupné z: https://whatis.techtarget.com/definition/front-end.

[36] Rouse, M. a Moore, L. *MySQL* [online]. TechTarget, Červenec 2018 [cit. 2020-04-7]. Dostupné z: https://searchoracle.techtarget.com/definition/MySQL.

[37] Saini, A. *Managed code and Unmanaged code in .NET* [online]. GeeksforGeeks, Duben 2019 [cit. 2020-02-20]. Dostupné z: https://www.geeksforgeeks.org/managed-code-and-unmanaged-code-in-net/.

[38] Satrapa, J. *Genealogie - Rodopis* [online]. Vysoká škola polytechnická Jihlava, Centrum celoživotního vzdělávání, 2016 [cit. 2020-03-06]. Dostupné z: https://www.vspj.cz/ISBN/Univerzita%20t%C5%99et%C3%ADho%20v%C4%9Bku%20-%20u%C4%8Debn%C3%AD%20materi%C3%A1ly/Genealogie%20-%20Rodopis%20-%20Jaroslav%20Satrapa.pdf.

[39] Schonning, N., Wagner, B., Pratt, T. et al. *What's new in .NET Framework* [online]. Microsoft, Duben 2019 [cit. 2020-02-23]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/whats-new/#v46.

[40] Singh, R. R. *Mastering Entity Framework*. 1. vyd. Packt Publishing Ltd., Únor 2015. ISBN 978-1-78439-100-3.

[41] Smola, M. *HTML5: co přináší a proč se o něj zajímat* [online]. Srpen 2011 [cit. 2020-03-05]. Dostupné z: https://www.root.cz/clanky/html5-co-prinasi-a-proc-se-o-nej-zajimat/.

[42] Stackify. *.NET Core vs .NET Framework: How to Pick a .NET Runtime for an Application* [online]. Září 2017 [cit. 2020-04-09]. Dostupné z: https://stackify.com/net-core-vs-net-framework/#wpautbox_about.

[43] TechTerms. *Web Application* [online]. TechTerms, Únor 2014 [cit. 2020-03-27]. Dostupné z: https://techterms.com/definition/web_application.

[44] TechTerms. *Client-Server Model* [online]. TechTerms, Červen 2016 [cit. 2020-03-27]. Dostupné z: https://techterms.com/definition/client-server_model.

[45] Tichopád, M. *Postman, 1. část* [online]. Kutáč, září 2020 [cit. 2020-03-16]. Dostupné z: https://www.kutac.cz/weby-a-vse-okolo/postman-1-cast.

[46] tutorialspoint. *TypeScript - Overview* [online]. tutorialspoint [cit. 2020-02-26]. Dostupné z: https://www.tutorialspoint.com/typescript/typescript_overview.htm.

[47] Vítek, M. *Distribuované systémy na platformě .NET Framework*. Brno, CZ, 2009. Dizertační práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/2959/Vitek_M_PhD_thesis.pdf?sequence=2&isAllowed=y.

[48] w3schools.com. *C# Introduction* [online]. w3schools.com, září 2013 [cit. 2020-02-25]. Dostupné z: https://www.w3schools.com/cs/cs_intro.asp.

[49] W^{3}Techs. *Usage statistics of markup languages for websites* [online]. 2020-03-02 [cit. 2020-03-02]. Dostupné z: https://w3techs.com/technologies/overview/markup_language.

[50] W^{3}Techs. *Usage statistics of server-side programming languages for websites* [online]. 2020-03-27 [cit. 2020-03-27]. Dostupné z: https://w3techs.com/technologies/overview/programming_language.

[51] Wenzel, M., Dykstra, T., Hoag, S. et al. *Common Language Runtime (CLR) overview* [online]. Microsoft, Duben 2019 [cit. 2020-02-19]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/clr.

[52] Wenzel, M., Dykstra, T., Latham, L. et al. *What is "managed code"?* [online]. Microsoft, Červen 2016 [cit. 2020-02-20]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/managed-code.

[53] Čermák, M. *Vícevrstvá architektura: tenký, tlustý a chytrý klient* [online]. Květen 2010 [cit. 2020-03-27]. Dostupné z: https://www.cleverandsmart.cz/vicevrstva-architektura-tenky-tlusty-a-chytry-klient/.

[54] Česká republika. Předpis 301/2000 Sb. In: *Zákon o matrikách, jménu a příjmení, částka 85*. Praha, Česká republika: Ministerstvo vnitra, únor 2003, s. 4107 – 4112. Sbírka zákonů 2003. Dostupné z: https://www.psp.cz/sqw/sbirka.sqw?cz=301&r=2000.

[55] Česká republika. Předpis 300/2006 Sb. In: *Vyhláška, kterou se provádí zákon č. 115/2006 Sb., o registrovaném partnerství a o změně některých souvisejících zákonů, a kterou se mění vyhláška č. 207/2001 Sb., kterou se provádí zákon č. 301/2000 Sb., o matrikách, jménu a příjmení a o změně některých souvisejících zákonů, ve znění pozdějších předpisů, částka 94*. Praha, Česká republika: Ministerstvo vnitra, červen 2006, s. 3719 – 3729. Sbírka zákonů 2006. Dostupné z: https://www.psp.cz/sqw/sbirka.sqw?cz=300&r=2006.

[56] Česká republika. *Nahlížení do matričních knih a sbírek listin* [online]. Ministerstvo vnitra, březen 2014 [cit. 2020-03-09]. Dostupné z: https://portal.gov.cz/obcan/zivotni-situace/obcan-a-stat/matriky/nahlizeni-do-matricnich-knih-a-sbirek-listin.html.

[57] Čápka, D. *Úvod do webových aplikací v ASP.NET* [online]. ITnetwork.cz, 2016 [cit. 2020-03-27]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/asp-dot-net-tutorial-uvod-do-webovych-aplikaci.

[58] Štorc, O. *Historie .NETu* [online]. ITnetwork.cz, 2015 [cit. 2020-02-20]. Dostupné z: https://www.itnetwork.cz/csharp/historie-dotnetu.

[59] Šípek, A. *Genealogie* [online]. 2014 [cit. 2020-03-07]. Dostupné z: http://www.genetika-biologie.cz/genealogie.

Příloha A

Instalace a spuštění

V této příloze bude popsán postup spuštění aplikace na lokálním webovém serveru. Premisou pro spuštění aplikace je přítomnost běhového prostředí Node.js, možnost založení MySQL databáze a server pro hostování ASP.NET Core aplikace. Níže je popsán proces spuštění aplikace na operačním systému Microsoft Windows 10.

Lokální spuštění na OS Microsoft Windows

Prvním krokem je založení MySQL databáze se všemi tabulkami, přičemž databáze musí nést jméno „familytreedb“. Do této databáze lze importovat data, která byla použita při testování (obsažena v souboru `database_export.sql`). Ke spuštění lokálního serveru pro databázi může být použita například aplikace XAMPP[1].

Ve druhé fázi je potřeba spustit server s webovou aplikací. Toho lze dosáhnout otevřením řešení *family-tree.sln* ve vývojovém prostředí Microsoft Visual Studio[2] a spuštěním serveru IIS Express. Předpokladem jsou nainstalované vývojářské nástroje .NET Core 3.1 SDK[3] a MySQL Connector[4].

Poslední vrstvou, kterou je nutné zprovoznit, je vrstva front-end. K běhu aplikace vytvořené aplikačním rámcem Angular je vyžadována přítomnost běhového prostředí Node.js[5]. Samotné spuštění lze provést z příkazové řádky použitím příkazu `ng serve` v příslušném adresáři. Pro komunikaci s aplikací DEMoS je při lokálním spuštění zapotřebí používat prohlížeč s vypnutou kontrolou mechanismu CORS.

Podrobnosti k demonstračním datům

Pro přístup k demonstračním datům je nutné se přihlásit patřičnými údaji. V aplikaci jsou vytvořeny dva uživatelské účty, kde přihlašovací jména jsou shodná s hesly. Přihlašovací údaje k účtům tedy jsou:

- uživatel Jan Svoboda – přihlašovací jméno: *svoboda*, heslo: *svoboda*
- uživatel Edvard Kovář – přihlašovací jméno: *kovar*, heslo: *kovar*

[1] XAMPP je ke stažení na: https://www.apachefriends.org/index.html
[2] Microsoft Visual Studio je ke stažení na: https://visualstudio.microsoft.com/cs/vs/
[3] .NET Core 3.1 SDK je ke stažení na: https://dotnet.microsoft.com/download/dotnet-core/3.1
[4] MySQL Connector je ke stažení na: https://dev.mysql.com/downloads/connector/net/
[5] Běhové prostředí Node.js je ke stažení na: https://nodejs.org/en/