
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# WEB NA PREPISOVANIE GENEALOGICKÝCH PRAMEŇOV

**A WEBSITE FOR TRANSCRIBING GENEALOGICAL SOURCES**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **JAKUB SOKOLÍK**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. JAROSLAV ROZMAN, Ph.D.**
**SUPERVISOR**



**BRNO 2023**

# Abstrakt

Systém DEMoS je vyvinutý na prepis genealogických údajov do webovej podoby a zároveň poskytuje možnosť vyhľadávania v týchto údajoch. V súčasnosti podporuje prepis a spracovanie 5 z 8 historických prameňov. Cieľom tejto práce je uskutočniť dôkladnú analýzu celého systému s cieľom identifikovať chýbajúce a nefunkčné časti. Následne sa zameriame na odstránenie týchto nedostatkov a doplnenie chýbajúcich častí, aby sme zabezpečili, že systém bude schopný plne fungovať v súlade s pôvodným návrhom. Jednou z hlavných úloh je doplnenie chýbajúcich historických prameňov, aby bolo možné pracovať zo všetkými 8 prameňmi v rámci systému. Okrem toho budeme venovať pozornosť nedostatkom v užívateľskom rozhraní a budeme sa usilovať o ich odstránenie. Toto zahŕňa optimalizáciu použiteľnosti a zvýšenie prehľadnosti systému.



# Abstract

The DEMoS system is developed to transcribe genealogical data into a web-based form and also provides the possibility to search this data. It currently supports the transcription and processing of 5 of the 8 historical sources. The aim of this work is to perform a thorough analysis of the whole system in order to identify missing and non-functional parts. We will then focus on correcting these deficiencies and filling in the missing parts to ensure that the system is able to fully function as originally designed. One of the main tasks is to complete the missing historical sources to be able to work with all 8 sources within the system. In addition, we will pay attention to the shortcomings in the user interface and work to address them. This includes optimizing usability and increasing the clarity of the system.



# Kľúčové slová

DEMoS, genealógia, prepis matričný záznamov, historické pramene, webová aplikácia, Latte, Nette, PHP, MySQL

# Keywords

DEMoS, genealogy, transcription of registry records, historical sources, web application, Latte, Nette, PHP, MySQL


# Citácia

SOKOLÍK, Jakub. *Web na prepisovanie genealogických prameňov*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

# Web na prepisovanie genealogických prameňov

## Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Jaroslava Rozmana, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jakub Sokolík
17. mája 2023


## Poďakovanie

Rád by som poďakoval vedúcemu práce Ing. Jaroslavi Rozmanovi, Ph.D. za odborné vedenie pri písaní tejto práce a za konzultácie.

# Obsah

# Zoznam obrázkov

# Kapitola 1

# Úvod

Genealógia alebo rodokmeňová veda má v súčasnosti stále väčšiu popularitu a záujem. Táto disciplína sa zaoberá skúmaním rodinných vzťahov a pôvodu jednotlivcov, čo je premnohých ľudí fascinujúce a zaujímavé. Informácie o svojich predkoch môžeme nájsť v rôznych zdrojoch ako sú napríklad matriky alebo rôzne kroniky. V minulosti bola práca s týmito prameňmi náročná, keďže si vyžadovala fyzickú návštevu archívov. S rozvojom informačných technológii sa tieto pramene stávajú dostupnejšie, a s tým aj stúpa záujem ľudí o dané informácie. Väčšina archívov je digitalizovaná, a tak je možné množstvo informácii dohľadať z pohodlia svojho domova.

Avšak digitalizácia archívov nie je odpoveď na všetko. V dokumentoch je stále potrebné pracne vyhľadávať. Po vyhľadaní niektorých informácií je však opäť nutné vrátiť sa k zdrojom ktoré sme už prezerali. Aby sa táto detektívna činnosť uľahčila a urýchlila vznikol v spolupráci Vysokého učenia technického v Brne a Masarykovej univerzity projekt DEMoS (Database of Early Modern Sources). Jedná sa o databázu genealogických prameňov s webovým rozhraním, kde dobrovoľníci môžu prepisovať informácie z historických prameňov. Nad takto prepísaniami dátami je potom možné jednoduché vyhľadávanie. Systém je navrhnutý tak, aby uchovával aj vzťahy medzi jednotlivými osobami. Pre užívateľa je potom oveľa jednoduchšie dohľadať rodinných príslušníkov a zostaviť tak svôj vlastný rodokmeň. Väčšina historických záznamov na území Českej republiky je písaná v staročeskom, nemeckom alebo latinskom jazyku. Fakt, že užívateľ môže využiť prepísaných záznamov bez nutnosti toho, aby tieto jazyky ovládal, môžeme zaradiť medzi prednosti systému. Využitie systému lajtskou verejnosťou kladie na systém vysoké nároky z hľadiska užívateľskej prívetivosti a spoľahlivosti celého systému. Fakt že vývoju systému sa venuje niekoľko bakalársky a diplomových práci, ktoré sa systém pokúšajú rozšíriť o novú funkcionalitu, vnáša do riešenia nekonzistenciu a chyby. Táto práca je zameraná na analýzu celého systému.

Cieľom je identifikovať chýbajúce a nefunkčné časti systému. Následne sa zameriame na odstránenie týchto nedostatkov a doplnenie chýbajúcich častí. Dôkladne sa pozrieme na jednotlivé časti systému a pokúsime sa identifikovať miesta, kde by v budúcnosti mohli vznikať chyby. Pokúsime sa takéto časti kódu eliminovať a do budúcnosti tak zaručiť konzistenciu systému. Systém rozšírime o chýbajúce historické pramene a doplníme dáta do databázy. Okrem toho budeme venovať pozornosť aj nedostatkom v užívateľskom rozhraní. Odstránením týchto nedostatkov sa pokúsime zvýšiť prehľadnosť systému a zabezpečiť, aby bol ľahko pochopiteľný a intuitívny.

# Kapitola 2

# Štúdium problematiky

V tejto kapitole sa zoznámime so základmi genealogie a podrobne si objasníme niektoré kľúčové pojmy, ktoré sú nesmierne dôležité pre správne pochopenie tejto problematiky. Budeme sa venovať aj historickým prameňom, ktoré nám poskytnú cenné informácie o štruktúre dát, s ktorými budeme neskôr pracovať. Osobitne sa budeme venovať tvorbe rodokmeňov. Spomenieme niektoré dostupné nástroje na tvorbu rodokmeňov.

## 2.1 Genealógia

Pojem genealógia pochádza z gréckeho slova genos (rod, pokolenie) a logos (náuka, veda). Je to vedná disciplína, ktorá sa zaoberá príbuzenskými vzťahmi medzi ľudskými jedincami, ktoré vychádzajú z ich spoločného rodového pôvodu.[19]. Počiatky tejto vedy siahajú do obdobia staroveku, starovekí Gréci vytvárali rodokmene svojich bohov a Egypťania poznali príbuzenské vzťahy medzi faraónmi.[23]

Genealogia je štúdium pôvodu, vzťahov, jednotlivcov a ich predkov. Je to disciplína, ktorá sa zaoberá výskumom rodokmeňov, zostavovaním rodinných stromov a zaznamenávaním genealogických údajov. Genealogia sa zaoberá nielen sledovaním biologických vzťahov, ale aj štúdiom sociálnych a kultúrnych aspektov, ktoré ovplyvňujú rodinné väzby, dedičnosť a kultúrne aspekty.

Výskumníci genealogie môži čerpať informácie z rôznych historický pramoňov, ktorými môžu byť napríklad rodné listy, manželské záznamy, cirkevné a náboženské záznamy, sčítania ľudu, historické dokumenty. Moderné technológie, ako sú digitálne archívy a genealogické databázy, umožňujú jednoduchší prístup k týmto informáciám.

## 2.2 Genealogické grafy

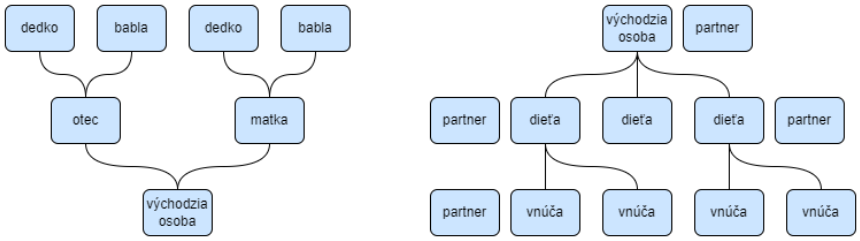
Pod pojmom rodokmeň si väčšinou predstavujeme nejaký stromový graf. Na začiatku je kmeň a ten sa potom vetví ďalej. štruktúra rodokmeňa sa skladá z jednotlivých osôb a vzájomných príbuzenských väzbách medzi nimi. Jedná sa tak o najjednoduchší genealogický model. Jednotlivý potomkovia sú zoradení od probanta[1] a to od minulosti až do prítomnosti. Rodokmene sa väčšinou zameriavajú na mužskú líniu a nachádza sa v nich minimum informácii o ženách. [22]

**Vývod** je najpoužívanejším genealogickým modelom, funguje na podobnom princípe ako rodokmeň s výnimkou toho, že uchováva informácie o mužskej aj ženskej línii. Často

[1] Probant je osoba z ktorej vychádza genealogický graf

sa zobrazuje vo forme stromu, kde je probant zobrazený ako kmeň a jeho predkovia sa postupne vetvia. [23]

**Rozvod** patrí medzi komplikovanejšie genealogické modely, jeho spracovanie je náročné. Probantom je väčšinou pár a graf znázorňuje všetkých priamych potomkov. Tento graf znázorňuje aj nemanželské deti a ich potomkov, čo prispieva k jeho zložitosti. [23]. Na obrázku 2.1 sú znázornené jednotlivé genealogické modely. Na ľavej strane je štruktúra vývod, na pravej strane je štruktúra rozvodu. Pátranie po predkoch je hlavnou motiváciou na vytvorenie systému DEMoS. Ten bude v neskoršej fáze schopný z prepísaných informácií z historických prameňov vytvárať rodokmeňe zo všetkých evidovaných osôb.




Obr. 2.1: Ukážky genealogických modelov

## 2.3 Tvorba rodokmeňu

Na tvorbu rodokmeňu je potrebné mať dostatok informácii o svojich predkoch. S prvotným bádaním môžeme začať už doma za využitia ústnych alebo písomných zdrojov. Na zostavenie prvého rodokmeňu postačí pero a papier, no je lepšie zvoliť nejaký online nástroj. Existuje množstvo nástrojov s užitočnými funciami, ktoré nám pri tvorbe rodokmeňu môžu pomôcť. Medzi obľúbené nástroje patrí MyHeritage alebo FamilySearch.

### MyHeritage

MyHeritage je populárna genealogická platforma, ktorá ponúka širokú škálu služieb a nástrojov pre výskum rodinných dejín. Užívatelia môžu vytvárať vlastné rodokmene, vyhľadávať historické záznamy, podstúpiť DNA testovanie, objavovať nové príbuzenstvo a zdieľať rodinné príbehy. S MyHeritage môžete objaviť svoj rodinný pôvod, pripojiť sa k rozsiahlej genealogickej komunite a hlbšie porozumieť svojim predkom. Táto služba poskytuje prístup k viac ako 16 miliardám historických záznamov. Je to cenný nástroj pre všetkých, ktorí majú záujem o genealógiu a štúdium rodinnej histórie. Základné služby sú dostupné bezplatne pre širokú verejnosť.

### FamilySearch

FamilySearch je internetová platforma poskytujúca širokú škálu služieb a zdrojov pre genealogický výskum a zaznamenávanie rodinných dejín. Je to bezplatná služba, ktorá umožňuje užívateľom vyhľadávať, zdieľať a spravovať genealogické údaje a dokumenty. FamilySearch ponúka prístup k veľkému množstvu historických záznamov z celého sveta, vrátane narodení, úmrtí, manželstiev, sčítaní ľudu a ďalších dôležitých udalostí. Tieto záznamy sú sprístupnené prostredníctvom online vyhľadávania, čo užívateľom umožňuje objavovať informácie o svojich predkoch a vytvárať rodinné stromy. Okrem toho poskytuje FamilySearch

aj nástroje na vytváranie a spravovanie rodokmeňov, ktoré umožňujú užívateľom zaznamenať a organizovať informácie o jednotlivých členoch svojej rodiny. K jednotlivým osobám v rodokmeni môžeme pridávať fotografie, príbehy a dokumenty. FamilySearch podporuje spoluprácu medzi užívateľmi, umožňuje im zdieľať svoje rodinné údaje a spolupracovať na vytváraní spoločných rodokmeňov. To umožňuje objavovať a prepojiť sa s novými príbuznými a rozšíriť výskum rodinného pôvodu.

## 2.4 Matriky

Matrika je najznámejší historický prameň, často využívaný pri genealogickom pátraní po predkoch. Ide o úradný dokument, ktorý uchováva záznamy o narodení, sobášoch a úmrtiach, a poskytuje cenné informácie o rodinách a ich vzájomných väzbách. Matriky sú pre genealógov neoceniteľným zdrojom, ktorý im umožňuje rekonštruovať rodinné dejiny a objavovať svoje korene.

V súčasnosti existujú dva druhy matrik. Živé matriky sú do neustále aktualizované novými záznamami, ktoré sa do nich pridávajú. Tieto matriky zostávajú uložené na matričných úradoch po dobu 100 rokov od posledného rodného záznamu alebo 75 rokov od posledného sobášneho či úmrtného záznamu. Po uplynutí tejto lehoty sa matriky považujú za neplatné a presúvajú sa do archívov, kde sú verejne prístupné. Tieto archívy poskytujú možnosť prehľadávať a študovať matriky.[23]


Obr. 2.2: Ukážky matričného záznamu z Moravsko zemského archívu

V roku 2007 sa v Českej republike začala masívna digitalizácia matrík, čo otvorilo nové možnosti pre genealogický výskum. Pracovníci jednotlivých archívov postupne skenujú všetky matriky a zverejňujú ich na webových stránkach. Tento proces digitalizácie má niekoľko výhod. Jednou z hlavných motivácií je uľahčenie prístupu k matrikám a umožnenie širokej verejnosti objavovať svoje rodinné korene. Okrem toho digitalizácia zabezpečuje dlhšiu životnosť originálnych matrík, pretože po digitalizácii nie je potrebné manipulovať s originálnou verziou.[1].

Aj keď digitalizácia matrík priniesla nové možnosti prístupu k informáciám, hľadanie v týchto historických prameňoch si stále vyžaduje určitú dávku trpezlivosti a odhodlania. Informácie v matrikách nie sú vždy jednoducho usporiadané alebo filtrované, čo môže predstavovať výzvu pri hľadaní konkrétnych záznamov. Ďalším faktorom, ktorý môže skomplikovať výskum, je skutočnosť, že staršie historické záznamy sú často písané v rôznych jazykoch a často ručne. Toto môže predstavovať výzvu pre výskumníkov, ktorí sa snažia získavať z nich relevantné informácie o svojich predkoch.

# Kapitola 3

# Systém DEMoS

V tejto kapitole sa detailnejšie zameriame na popis technológií, ktoré sú využívané v súčasnom systéme DEMoS. Hoci sme túto aplikáciu nevytvorili sami, prostredníctvom jej štúdia sme nadobudli dostatočné poznatky a skúsenosti na to, aby sme mohli zhodnotiť jej aktuálny stav. Okrem toho sa budeme venovať identifikácii potenciálnych chýb a nedostatkov, ktoré sme pri analýze zistili. Cieľom je poskytnúť komplexný pohľad na fungovanie tohto systému a navrhnúť možné zlepšenia.

## 3.1 Použité technológie

V tejto časti popíšeme jednotlivé technológie, ich zmysel a použitie v aplikácii. Nakoľko systém prechádzal početnými úpravami, jedná sa o dôležitú časť práce, ktorá popisuje aktuálny stav systému. Väčšina technológii popísaných v tejto kapitole bola popísaná v bakalárskej práci Davida Czernína,[15] Nakoľko projekt prešiel mnohými zmenami, zhrnieme si jednotlivé technológie znovu, tak aby odpovedali súčasnému stavu aplikácie DEMos.

### 3.1.1 PHP

Ako základ aplikácie je použitý programovací jazyk PHP. Tento jazyk bol vybraný vzhľadom na jeho rozšírenosť, veľké množstvo dostupných komponent a vhodnosť pre webové aplikácie.[15].

PHP, z anglického Hypertext Preprocessor, je open-source skriptovací programovací jazyk používaný na strane servera. Prvá verzia tohto jazyka bola spustená v roku 1995. Je určený predovšetkým na tvorbu dynamických webových stránok. Tento jazyk používa množstvo etablovaných spoločností a technologických gigantov ako sú napríklad Facebook, Wikipedia, Spotify alebo WordPress.[14] PHP ponúka flexibilný prístup k objektovo-orientovanému programovaniu, čím umožňuje vytvárať opakovane použiteľné a komplexné webové aplikácie.[25] Najnovšou verziou je verzia 8.2 ktorá bola vydaná v novembri roku 2022. Momentálne najpoužívanejšou verziou tohto jazyka je verzia 8.1. Systém DEMoS využíva verziu 7.4, ktorá je druhou najpoužívanejšiu verziou tohto programovacieho jazyka.[13]

### 3.1.2 Composer

Composer je open source nástroj na správu závislostí pre PHP, vytvorený predovšetkým na uľahčenie distribúcie a údržby balíkov PHP ako jednotlivých komponentov aplikácie. Umožňuje jednoduchým spôsobom inštalovať a aktualizovať knižnice jazyka PHP. Taktiež

dokáže jednoduchým spôsobom udržiavať závislosti na verziách týchto knižníc.[18] Inštalácia Composera je jednoduchá, a v závislosti na použitom operačnom systéme stačí použiť balikovácí nástroj apt-get pre Linux alebo Brew pre MacOs. Inštalátor pre Windows je možné stiahnuť na oficiálnych stránkach Composeru.[9]. Použité knižnice sa následne definujú v súbore *composer.json*, ktorý okrem iného obsahuje informácie o jednotlivých verziách použitých knižníc.

### 3.1.3 Framework Nette

Pre jazyk PHP existuje niekoľko frameworkov, ktoré slúžia ako základ pre Váš projekt. Framework je v podstate sada funkcií, ktoré uľahčujú život programátorovi. Funkcie vytvárajú štruktúru projektu, kde sú spracované takmer všetky všeobecné úlohy a tak nie je potrebné sa zo začiatku zaoberať vytváraním zbytočne logiky. Vďaka tomu, že framework stanovuje základnú štruktúru projektu, členovia vývojárskeho tímu majú jasnejší obraz o tom, ako by mali pristupovať k rôznym častiam projektu. To znižuje riziko chýb a zaručuje, že kód bude jednotný a prehľadný. Toto všetko spolu prispieva k lepšej údržbe a rýchlejšiemu vývoju aplikácie.[21] Medzi známe PHP frameworky patrý Nette, Laraver alebo Symfony.

Pre systém DEMoS bol zvolený framework Nette, pretože má najširšie pole pôsobnosti a je najpoužívanejším frameworkom v Českej republike.[15] Autorom tohto frameworku je Čech David Grudl, a aj preto je množstvo dokumentácie a fórum písané práve v českom jazyku.[4]

#### Architektúra Nette Frameworku

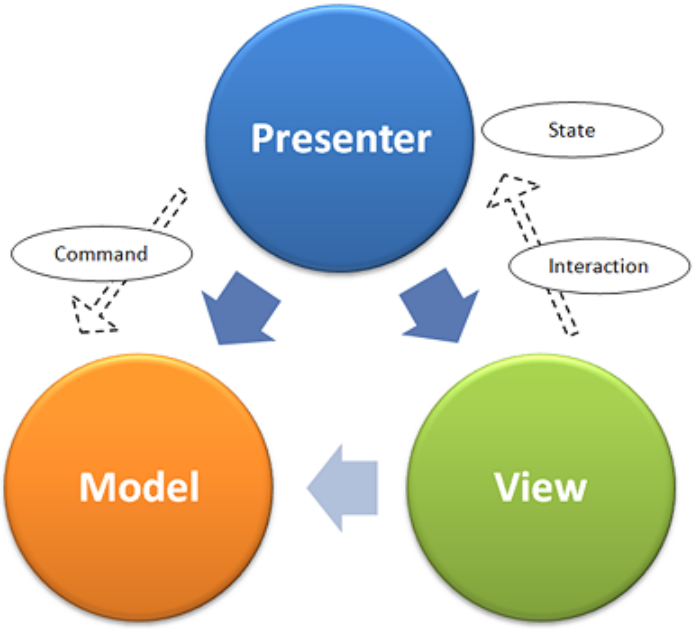
Framework Nette využíva softvérovú architektúru MVC. Skratka architektúry opisuje jej 3 základne komponenty v tomto poradí: model, view a controller.

Architektúra MVC rozdeľuje celú aplikáciu na dátový model aplikácie, užívateľské rozhranie a riadiacu logiky do už spomenutých 3 komponent. Toto delenie umožňuje oddelenie rôznych vrstiev aplikácie a ich nezávislé vývoj. Každá vrstva svoju špecifickú úlohu, a je tak možné sa sústrediť na vývoj konkrétnej vrstvy bez potreby riešenia ostatných. Takýmto spôsobom sa zvyšuje modularita a flexibilita aplikácie, čo má vplyv na následnú údržbu alebo rozširovanie funkcionality.[17] Predstavme si jednotlivé komponenty tohto návrhu a pozrime sa na ich úlohu v architektúre MVC. Použitie architektúry MVC v Nette aplikácii je znázornené na obrázku 3.1.

- **Model** zabezpečuje správu a manipuláciu s dátami aplikácie. Jeho úlohou je vykonávať operácie nad dátami a poskytovať rozhranie pre prístup k týmto údajom controlleru a zobrazovacej vrstve (view). Týmto spôsobom model umožňuje oddelenie biznis logiky aplikácie od jej prezentácie a interakcie s používateľom.

- **View** alebo zobrazovacia vrstva, slúži na prezentáciu dát a informácií používateľovi, a to zvyčajne vo forme vizuálneho rozhrania. Okrem iného poskytuje používateľovi interakciu s aplikáciou, môže obsahovať prvky, ako sú tlačidlá, polia na vstup dát, grafy, tabuľky a ďalšie. Jeho úlohou je zabezpečiť, aby používateľ videl a mohol používať dáta z modelu, pričom všetky manipulácie s nimi sú zabezpečené pomocou controlleru. Vo väčšine frameworkov, je view reprezentovaný šablonovácim systémom. Nette používa šablónovací systém Latte (3.1.3).

- **Controller** slúži na spracovanie užívateľských požiadaviek, ktoré prichádzajú zo zobrazovacej vrstvy. Controller prijíma vstup od užívateľa, riadi tok aplikácie, určuje




Obr. 3.1: Architektúrá MVC v Nette aplikácií[17]

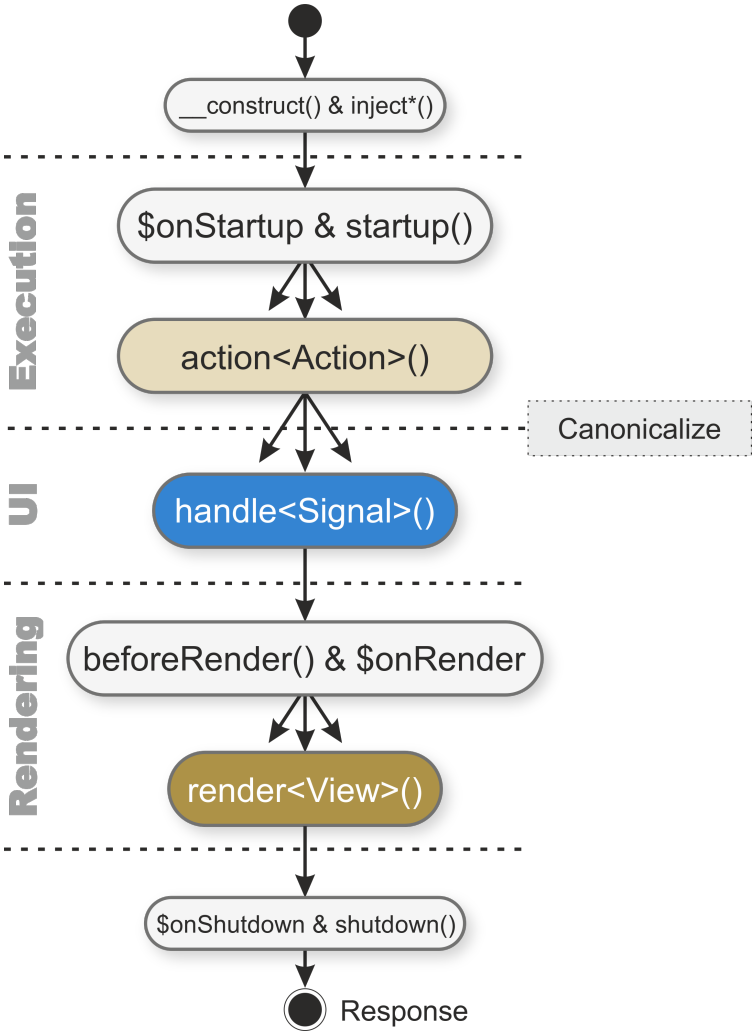
ktorý model bude použitý a aká zobrazovacia vrstva (view) bude použitá na základe akcie, ktorú užívateľ vykonal. Controller tak zabezpečuje oddelenie jednotlivej funkcionality a zvyšuje modularitu a znovupoužiteľnosť kódu v aplikácii. V Nette aplikácii je controller reprezentovaný ako presenter. Životný cyklus presenteru je zobrazený na obrazku 3.2

### Dependency Injection

Dependency Injection (skrátene DI) je návrhový vzor, ktorý umožňuje vkladať závislosti do objektu prostredníctvom konštruktora alebo metód, namiesto toho, aby si objekt sám vytváral svoje závislosti. Týmto spôsobom sa znižuje viazanosť medzi jednotlivými objektami. Dependency Injection je dôležitým prvkom v moderných softvérových aplikáciách a pomáha vytvárať flexibilné a udržateľné systémy.[16]

Základom takmer všetkých projektov postavených na Nette frameworku je staticky dependency injection kontajner. V tomto kontajnery sa uchovávajú registrácie všetkých komponentov, služieb, objektov a parametrov ktoré aplikácia využíva. Medzi použité komponenty, ktoré je treba registrovať patrí napríklad knižnica doctrine (3.1.3) Kontajner sa vygeneruje automaticky z konfiguračného súboru *config.neon*.[6] Tento konfiguračný súbor je písaný v jazyku NEON. Jedná sa o ľudsky čitateľný formát štruktúrovaných dát, používaný prevažne v Nette aplikáciach. Vygenerovaný kontajner sa ukladá vo forme PHP kódu do cache[1]. Nette aplikácie si cachované údaje ukladajú do zložky temp.

[1]Dáta ktoré sa často používajú a je zložité ich vygenerovať sa raz vygenerujú a uložia pre ďalšie použitie. Takýmto spôsobom je možné urýchliť načítanie aplikácie, nakoľko dáta nie je treba zakaždým pracne generovať




Obr. 3.2: Životný cyklus Nette presenteru[8]

### Generované továrne

Generované továrne (anglicky generated factories) sú v Nette frameworku triedy, ktoré majú za úlohu vytvárať a konfigurovať objekty. Ich generovanie je založené na princípe Dependency Injection (3.1.3). Továrne slúžia na inštancovanie a správu objektov v aplikácii a výrazne uľahčujú proces vytvárania objektov a minimalizujú chyby pri konfigurácii závislostí. Týmto spôsobom je zabezpečená flexibilita a znovupoužiteľnosť kódu.

Generované továrne majú dôležitú úlohu v efektívnom riadení objektov v Nette frameworku. Sú schopné vytvárať a spravovať závislosti objektov automaticky, čo ušetrí mnoho času a zjednoduší proces inicializácie objektov. Továrne zabezpečujú správne nastavenie závislostí a ich automatickú injekciu do objektov, čím minimalizujú riziko vzniku chýb v kóde. [7]

### Šablonovací systém Latte

Latte je šablónovací systém používaný v Nette Frameworku. Ako už bolo spomenuté v architektúre Nette frameworku 3.1.3 jedná sa o zobrazovacia vrstva (view) architektúry MVC.

Latte je navrhnuté tak, aby umožnil jednoduché oddelenie prezentácie a logiky výstupu. Latte umožňuje vkladať do šablón premenné, podmienky, cykly, definície funkcií, a taktiež rôzne zástupné znaky, ako napríklad náhradu HTML entít. Ďalším z prínosov šablónovacieho systému sú mechanizmy opätovného použitia a dedičnosti šablón. Tieto mechanizmy zvýšia vašu produktivitu, pretože každá šablóna obsahuje iba svoj jedinečný obsah a opakované prvky a štruktúry sa znovupoužijú.

Šablóny sú kompilované do PHP kódu, čo zaručuje ich rýchlosť a bezpečnosť. Bezpečnosť je zabezpečená pomocou rôznych funkcií a filtrov ktoré Latte využíva. Automatické escapovanie, ktoré chráni pred vložením nebezpečného kódu do šablóny, je efektívnou ochranou proti kritickoej zraniteľnosti - Cross-site Scripting (XSS)[2].[3]

### Tracy

Tracy je výkonná open source knižnica pre PHP, zameraná na ladenie, logovanie a zobrazovanie chýb v PHP aplikáciách. Poskytuje širokú škálu užitočných funkcií, ktoré prispievajú k efektívnemu vývoju aplikácií.

Jednou z hlavných výhod Tracy je schopnosť poskytovať podrobné výpisy chýb, ktoré obsahujú informácie o mieste v kóde, kde k chybe došlo. Tieto výpisy obsahujú aj ukazovatele, ktoré umožňujú presne identifikovať príslušnú časť kódu. Okrem toho, Tracy umožňuje sledovať jednotlivé volania funkcií, meranie výkonu, monitorovanie dotazov do databázy a aj sledovanie ajaxovej[3]

Tracy je tiež ľahko rozšíriteľný vďaka rôznym dostupným doplnkom a modulom. Táto flexibilita umožňuje vývojárom prispôsobiť si nástroje podľa vlastných potrieb a požiadaviek. Okrem toho, Tracy poskytuje užívateľsky prívetivé rozhranie, ktoré zjednodušuje prácu s nástrojmi a zvyšuje efektivitu pri ladení a opravovaní chýb v aplikáciách.

Tracy je nezávislá knižnica a nie je viazaná na Nette framework. Je možné ju ľahko nainštalovať do akéhokoľvek PHP projektu pomocou Composeru (3.1.2), čo umožňuje využívať jej výhody bez nutnosti použitia celého Nette frameworku.

### Doctrine

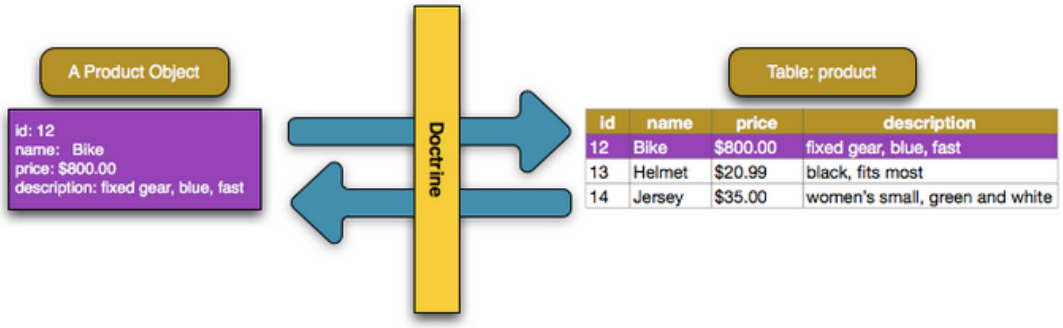
Doctrine je rozsiahla ORM (Object-Relational Mapping) knižnica určená pre programovací jazyk PHP. Vďaka tejto knižnici je možné jednoducho a efektívne pracovať s relačnými databázami. Jedným z hlavných prínosov Doctrine je jej schopnosť mapovať dáta z relačnej databázy na objekty, čo uľahčuje manipuláciu s nimi.[2]

Hlavným cieľom Doctrine je zjednodušiť a optimalizovať prácu s databázou. Vďaka jej funkciám a nástrojom je možné pohodlne definovať vzťahy medzi tabuľkami v relačnej databáze a ich mapovanie na objekty. Týmto spôsobom sa minimalizujú chyby a zjednodušuje sa proces práce s dátami. Znázornenie mapovania medzi objektami a tabuľkami je možné nájsť na obrázku 3.3, ktorý vizualizuje, ako Doctrine spracováva a interpretuje relačné dáta v objektovom modeli.

Doctrine je flexibilná a podporuje rôzne typy relačných databáz ako MySQL, PostgreSQL, NoSQL alebo MongoDB. Vďaka tejto podpore je možné efektívne integrovať Doctrine do rôznych projektov a pracovať s rôznymi typmi databáz.

[2] XSS je jednou z najčastejších zraniteľností webových stránok. Umožní útočníkovi vložiť do cudzej stránky škodlivý skript, ktorý sa spustí v prehliadači.

[3] AJAX (Asynchronous JavaScript and XML) označuje technológiu používanú na vytváranie asynchrónnych webových aplikácií. Tie sa od bežných webov líšia tým, že si vedia načítať dáta na pozadí, bez nutnosti znova nahrávať celú stránku.[24]




Obr. 3.3: Mapovanie entity na tabuľku v DB pomocou doctrine[2]

### Console

Console je knižnica pre PHP, ktorá umožňuje jednoduché a interaktívne používanie konzoly v aplikácii. V projekte je použitý balíček *kdyby/console*, ktorý je určený priamo pre Nette framework. Umožňuje vytvárať príkazy na správu a administráciu aplikácií priamo z príkazového riadku. Najčastejšie ša používa spolu s cronom[4], ktorý umožňuje plánovať spúšťanie skriptov potrebných pre chod aplikácie (napríklad odosielanie emailov alebo údržbu databáze). Svoje využitie nájde aj pri jednorazových udalostiach, napríklad vkladanie dát do databáze. Knižnica ponúka množstvo užitočných funkcií, ako napríklad generovanie nápovedy pre príkazy, generovanie progressBaru, správu argumentov alebo volieb príkazov.

### 3.1.4 jQuery

jQuery je rýchla, malá a na funkcie bohatá knižnica jazyka JavaScript, jedná sa o frontendovú knižnicu[5].jQuery je určený na zjednodušenie a zrýchlenie vývoja interaktívnych webových stránok. Umožňuje rýchlo a jednoducho pracovať s DOM (Document Object Model) stromom, teda so štruktúrou stránky a jej elementami, ako sú napr. tlačidlá, formuláre, obrázky atď.

S pomocou jQuery je možné jednoducho získať hodnoty vstupov od používateľov, vytvárať animácie a efekty, zobrazovať a skrývať prvky na stránke, zabezpečovať asynchrónne načítanie dát zo servera, pracovať s udalosťami ako kliknutie, scrollovania alebo odoslanie formulára.

jQuery je jednou z najpoužívanejších knižníc pre webové aplikácie a umožňuje vývojárom ušetriť veľa času a námahy pri implementácii rôznych interaktívnych prvkov a funkcií na webovej stránke. jQuery pokračuje v napĺňaní hesla svojich vývojárov: „Píšte menej, robte viac.“[12]

### 3.1.5 Bootstrap

Bootstrap je open-source framework pre vývoj responzívnych webových stránok a aplikácií. Mark Otto a Jacob Thornton vyvinuli Bootstrap pre Twitter s cieľom zlepšiť konzistentnosť nástrojov používaných na stránke a znížiť údržbu. Je založený na kombinácii HTML, CSS

[4]cron - jedná sa o systém plánovania úloh v operačnom systéme Linux

[5]Frontend označuje časť webovej aplikácie, ktorú používatelia vidia a interagujú s ňou priamo v prehliadači

a JavaScriptu, a poskytuje širokú škálu preddefinovaných štýlov, komponentov a nástrojov, ktoré zjednodušujú proces vývoja a zlepšujú užívateľské rozhranie. Vďaka svojej jednoduchosti a rozsiahlym možnostiam je Bootstrap veľmi populárnym nástrojom pre vývoj webových aplikácií a umožňuje rýchlejší a efektívnejší vývoj webových rozhraní s profesionálnym vzhľadom.[28]

## 3.2 Štruktúra projektu

Štruktúra projektu vychádza zo základného projektu Nette[5], ktorý bol nainštalovaný pomocou nástroja Composer(3.1.2).[15] Ďalšie komponenty využite v projekte, ktoré sú popísané vyššie v tejto kapitole, boli taktiež nainštalovane pomocou Composera.

Vnútorná štruktúra adresárov a jednotlivých zdrojových kódov je nasledovná:

- **app** obsahuje aplikačnú logiku.
  - **config** - obsahuje konfiguračné súbory potrebné pre samotný beh Nette aplikácie, a použitých komponent (3.1.3)
  - **forms** - adresár obsahuje triedy upravujúce vykresľovanie formulárov oproti preddefinovanému zobrazeniu Nette
  - **model** - adresár v ktorom sú umiestnené komponenty dátového modelu architektúry MVC (3.1.3)
  - **presenters** - adresár v ktorom sú umiestené jednotlive controllery architektúry MVC
    - **templates** - adresár obsahuje šablony Latte, tie sú rozdelené do jednotlivých podadresárov prislúchajúcim konkrétnemu presenteru. Táto štruktúra uľahčuje orientáciu v jednotlivých Latte šablonách
  - **router** - je súčasťou Nette frameworku a slúži k dynamickému generovaniu URL adresy
  - **enums** - adresár združuje všetky enumy použité v aplikácií
  - **console** - adresár obsahuje všetky skripty spustiteľné v príkazovom riadku pomocou komponenty Console (3.1.3)
  - **components** - obsahuje formulárové prvky použité v aplikácii
  - **factory** - obsahuje kódy pre Generované továrne (3.1.3) frameworku Nette
  - **vendor** - obsahuje zdrojové kódy jednotlivých komponent nainštalovaný prostredníctvom Composeru (3.1.2)
- **bin** - obsahuje súbor *console.php*, ktorý slúži na spúšťanie skriptov určených pre Console (3.1.3)
- **log** - adresár uchováva informácie určené k ladeniu aplikácie. Tieto informácie sú generované automaticky za behu aplikácie pomocou komponenty Tracy (3.1.3)
- **temp** - uchováva dočasné súbory aplikácie, cachované dáta
- **www**
  - **css** - obsahuje súbory so štýlmi CSS

- **images** - obsahuje jednotlivé obrázky použité vo webovej aplikácii
- **js** - obsahuje JS súbory
- **data** - obsahuje dáta vo formáte JSON. Jedná sa predovšetkým o dáta k jednotlivým archívom. Použitie týchto dát sa venujeme v kapitole 4.1

## 3.3 Historické pramene

Celý systém DEMoS je postavený na prepise jednotlivých záznamov, ktoré sú naskenované jednotlivými archami a verejne dostupné. Systém je navrhnutý tak, aby bolo možné prepisovať rôzne historické prameňe. Nakoľko jednotlivé historické pramene uchovávajú rôzne druhy informácií je pre nich v systéme vytvorený zakaždým osobitný model a presenter. Jedná sa o týchto 8 prameňov:

- **Berni rula** - jedná sa o historický súpis daňových povinností v Českom kráľovstve od 17 storočia. V systéme sa pracuje s anglickým prekladom tohto názvu **TaxGneiss**
- **Lánové rejstříky** - sú obdobou katastrov poddanských usadlostí a pozemkov na území Moravy. V systéme sa pracuje s anglickým prekladom tohto názvu **LandIndex**
- **Matriky** - sú zdrojom údajov o obyvateľstve, typicky o narodeniach, sobášoch a úmrtiach. V systéme sa pracuje s anglickým prekladom tohto názvu **Register**
- **Poddannská přiznávací fase** - sú najdôležitejšou súčasťou rektifikačných akt, ktoré sa vzťahujú na územie Moravy a obsahujú dôležité údaje o pozemkoch, ich majiteľoch, výmerách polí alebo vinníc. V systéme sa pracuje s anglickým prekladom tohto názvu **Phase**
- **Pozemkové knihy** - jedná sa o úradne zoznamy, do ktorých sa zapisovali nehnuteľnosti a určité práva, ktoré sa k ním sťahovali. V systéme sa pracuje s anglickým prekladom tohto názvu **LandBook**
- **Sčítací operáty** - sú to záznamy o sčítani ľudu. V systéme sa pracuje s anglickým prekladom tohto názvu **Census**
- **Urbáře** - sú historické súpisy povinností poddaných voči vrchnosti. V systéme sa pracuje s anglickým prekladom tohto názvu **Urbarium**
- **Súpis poddaných dle víry** - jedná sa o prvý pokus štátnej moci spísať celé obyvateľstvo v Čechách. V systéme sa pracuje s anglickým prekladom tohto názvu **SubjectList**

### 3.3.1 Uloženie historických prameňov v databáze

Na obrázku 3.4 je znázornené uloženie historických prameňov v databáze. Hlavné údaje sú uložené v tabuľke *register* a odlišuje ich stĺpec type. Stĺpec type je enum a pre jednotlivé pramene nadobúda následujúce hodnoty:

- **MAT** pre matriky
- **CEN** pre sčítací operáty

- **LR** pre lánové rejstříky
- **RA** pre poddannskú přiznávací fasi
- **URB** pre urbáre a **URBOP** pre opisy urbárov
- **BR** pre berní rulu
- **LBO** pre pozemkové knihy
- **SPPV** pre súpis poddaných dle víry

Jazyky, ktoré sa vyskytujú v jednotlivých prameňoch sú reprezentované lang enumom, ktorý môže podla normy ISO-639-1 nadobúdať hodnoty: CZ, GE, PL, SK, FR, HE, RU, EN. Okrem týchto enum obsahuje označenie LAT pre latinčinu, SC pre staroslovenčinu a NONE pre prázdnu hodnotu. Tieto jazyky sú použité aj v ostatných tabuľkách, ktoré sa v systéme vyskytujú.

Radenie jednotlivých záznamov je podla obce je uložené v tabuľke *unicipality*, v ktorej sú uložené jednotlivé obce previazané na RÚIAN[6]. Na túto tabuľku je následne naviazaná celá hierarchia tabuliek, ktorá reprezentuje územné rozdelenie od štátu až po časť obce.

Nakoľko sa jednotlivé pramene zapisujú na papierové médium, vždy musí byť uchovaná informácia o časovom rozpätí, ktoré ho sa daný historický prameň týka. Kvôli tomu, že v jednej matrike môžu byť údaje o narodeniach, úmrtiach a sobášoch v inom časovom období. Jedná sa tak o reláciu jedna ku n (anglicky One-To-Many), čiže údaje musia byť uložené v separé tabuľke - *registerrange*. Range type v tomto prípade nadobúda následujúce hodnoty: common, birth, death, marriage, indxBirth, indxDeath, indxMarriage.

Územné pokrytie jednotlivých záznamov sú uchované v tabuľke *register_municip*. Pokiaľ sú k danému historickému prameňu dostupné skeny jednotlivých strán dostupné online, url adresa k danému skenu vzniká zložením stĺpca scan_baseaddr z tabuľky *register* a stĺpcom url z tabuľky *registerscan*.

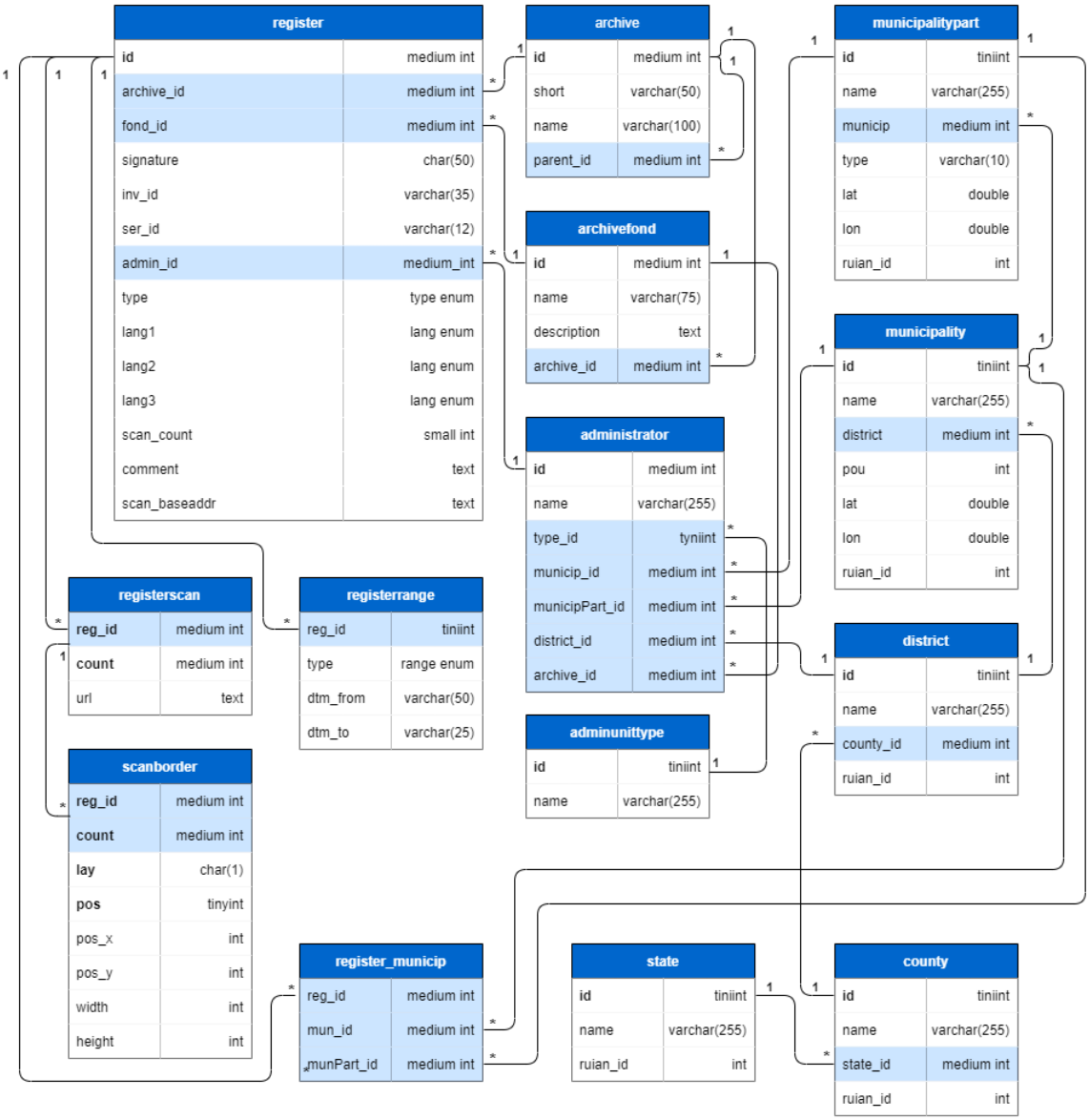
## 3.4 Súčasný stav aplikácie

Táto sekcia zameriava svoju pozornosť na dôkladné preskúmanie a testovanie súčasného stavu s cieľom poskytnúť komplexnú analýzu systému DEMoS. Na základe tejto analýzy budú navrhnuté postupy pre ďalší postup práce. V nasledujúcej časti tejto práce je definovaní cieľ práce. V následujúcich častiach sa detailne popisujú zistené nedostatky v užívateľskom rozhraní, ako aj chýbajúca funkcionalita a absencia dát. Taktiež sa snažíme objasniť aké dopady majú jednotlivé nedostatky na systém. Týmto podrobným pohľadom sa usilujeme poskytnúť komplexný prehľad súčasného stavu a identifikovať oblasti, kde je potrebné zlepšiť.

### 3.4.1 Cieľ prace

Cieľom tejto práce je podrobne analyzovať systém DEMoS ako celok a identifikovať jeho nedostatky a nefunkčné časti. Na základe tejto analýzy sa budeme sústrediť na odstránenie týchto nedostatkov, aby sme systém pripravili na efektívne používanie. Jedným z hlavných

[6]RÚIAN je skratka pre Registre Územnej Identifikácie, Adries a Nemovitostí. Je to informačný systém, ktorý spravuje a poskytuje údaje o parcelách, budovách, nehnuteľnostiach a adresách na území Českej republiky.




Obr. 3.4: Databáza pre uloženie jednotlivých archivačných prameňov

úloh bude doplnenie chýbajúcich historických prameňov, aby systém dokázal spracovať a podporovať všetkých 8 prameňov. Okrem toho budeme venovať pozornosť nedostatkom v užívateľskom rozhraní a usilovať sa o ich zlepšenie. Týmto komplexným prístupom sa snažíme zabezpečiť, aby systém DEMoS fungoval spoľahlivo a poskytoval optimálne užívateľské skúsenosti.

### 3.4.2 Nedostatky užívateľského rozhrania

Užívateľské rozhranie je kľúčovým prvkom akéhokoľvek softvérového systému, pretože je to hlavný bod kontaktu medzi užívateľom a aplikáciou. Ak je rozhranie neprehľadné, zložité alebo neresponzívne, môže to mať vplyv na používateľskú skúsenosť a zhoršiť efektivitu používania systému. Užívatelia sa môžu cítiť frustrovaní, môžu potrebovať viac času na dokončenie úloh, alebo sa môžu dokonca rozhodnúť nepoužívať systém vôbec.

Preto je dôležité venovať sa užívateľskému rozhraniu a zabezpečiť, aby bolo používateľsky prívetivé, jednoduché na ovládanie a efektívne v používaní. To môže zvýšiť produktivitu používateľov, zlepšiť ich spokojnosť a zabezpečiť, že systém bude používaný v plnej miere.

Po preskúmaní a otestovaní už existujúceho systému boli zistené následujúce nedostatky používateľského rozhrania:

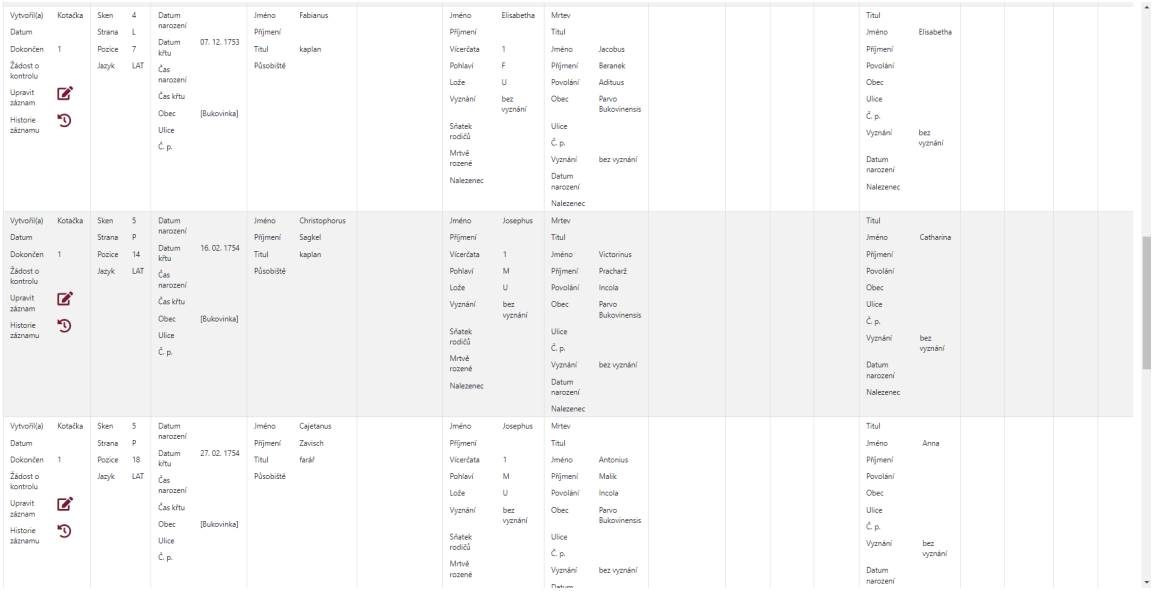
- tabuľky s prepísanými záznamami sú obšírne a zobrazujú veľké množstvo informácii. Tabuľky pretekajú ako po x-sovej, tak aj po y-novej, čo značne stážuje orientáciu v tabuľkách. Náhľad do tabuľky tak ako ho vidí užívateľ na desktopovom zariadení[7] je na obrázku 3.5. Pre uľahčenie orientácie v tabuľkách bude treba vytvoriť sticky[8] záhlavie tabuľky, aby užívateľ vedel, na ktorú časť tabuľky sa pozerá. Uľahčiť orientáciu v tabuľke je možné aj tým, že užívateľovi umožníme schovať stĺpce tabuľky, ktoré ho nezaujímajú. Riešeniu tohto problému sa venuje kapitola 4.2.

- Responzibilita užívateľského rozhrania pri prepise jednotných záznamov je chybne nastavená. Pri zariadeniach so šírkou 1300px až 1650px dochádza buď k prekrývaniu jednotlivých komponent na stránke alebo k ich neprirodzenému zarovnaniu. Daný rozsah je typický pre desktopové zariadenia, od veľkosti laptopu po stolový počitač. Tento stav je znázornený na obrázku 3.6. Pre bežného užívateľa môže byť daná chyba zmätočná a v mnohých prípadoch môže viesť k rozhodnutiu systém znovu nepoužiť. Ďalším miestom, kde sa chyba s responsibilitou vyskytuje je stránka s prepísanými záznamami, kde pri rozlíšení mobilného telefónu (okolo 450px až 800px) dochádza k preusporiadaniu hlavičky s údajmi o matrike. Stav je znázornený na obrázku 3.7. Odstránenie problémov s responsibilitou sa venuje kapitola 4.3.

- Pri prepise jednotlivých záznamov, je formulár rozdelený do sekcií. Jednotlivé sekcie sú usporiadané do 3 stĺpcov, tak ako je to vidieť na obrázku 3.7. V spodnej časti stránky sa nachádzajú tlačidlá na uloženie formulára, ktoré sú pri prepise matričných záznamov zarovnané na stred stránky. Pri prepise ostatných záznamov z iných historických prameňov sú však tieto tlačidlá zarovnané, rovnako ako sekcie formulára, do posledného stĺpca. Okrem iného majú aj neštandardné orámovanie, tak ako jednotlivé sekcie. Táto chyba je spôsobená tým, že sú vytvorené css štýly pre tlačidlá

[7] medzi desktopové zariadenia patrí každe zariadenie so šírkou zobrazenia 1200px alebo viac. Hranica je orientačná môže sa mierne meniť vzhľadom na návrh užívateľského rozhrania aplikácie

[8] Vlastnosť sticky v CSS sa používa na umiestnenie prvku na konkrétne miesto na obrazovke pri scrollovaní stránky.



Obr. 3.5: Tabuľka s prepísanými záznamami

v matričných záznamoch, ale pre ostatné pramene tieto štýly chýbajú. Nakoľko sa zakaždým jedná o rovnaké usporiadanie, premenovali sme triedu *addBirthForm-buttons* na *addForm-buttons* a použili ju vo všetkých formulároch.

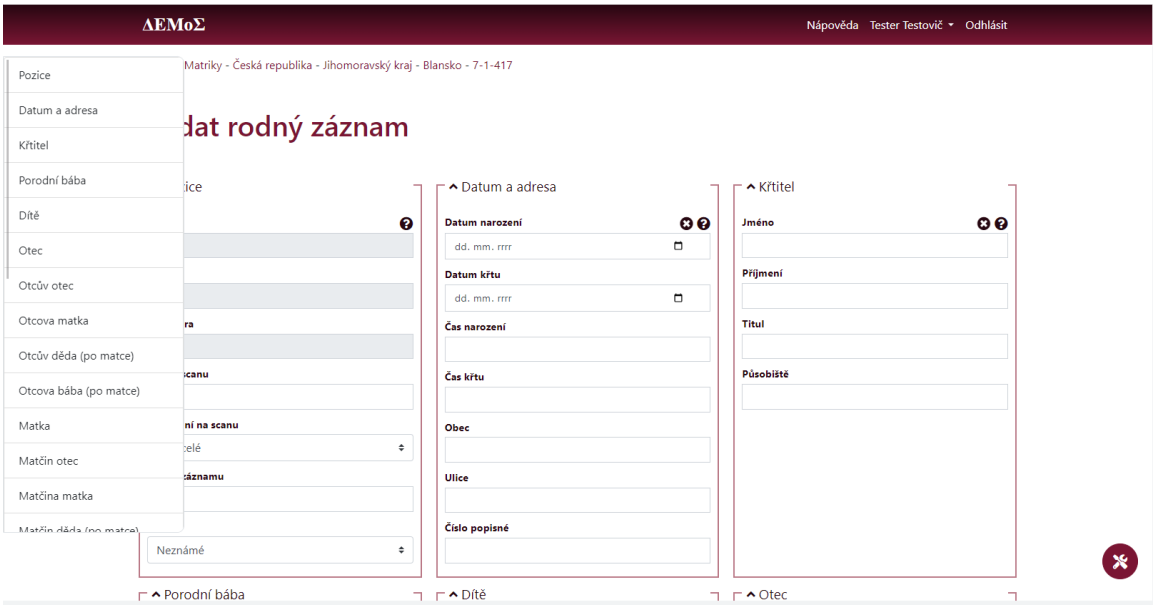
- Vyhľadávanie jednotlivých historických prameňoch na základe signatúry je doplnené o autocomplete search[9]. Návrhy, ktoré aplikácia ponúka užívateľovi nie sú zoradené, a nie sú ani anotované. Pri vyhľadávaní matriky so signatúrou 417, ktorá v systéme určí je, by sme očakávali, že práve tento návrh bude prvý, systém nám však takúto matriku vôbec neponúkne. Okrem toho, by bolo vhodné užívateľovi okrem signatúry poskytnúť aj iné informácie, ako napríklad kraj alebo číslo archívu aby užívateľ vedel, že sa jedná o matriku, ktorú naozaj hľadá. Aktuálny nevyhovujúci stav je zobrazený na obrázku 3.8. Vytváraniu týchto návrhov sa bude venovať samostatná kapitola 4.4.

- Pri vyplňovaní nového záznamu je užívateľovi automaticky ponúkaná normalizácia. Polia určené na štandardizáciu sú schované a zobrazujú sa až v prípade, keď užívateľ dané pole vyplní. Táto funkcionalita však nefunguje v prípade prepisu rodných záznamov nefunguje pre údaje o kmotroch. V prípade prepisu úmrtných a sobášnych záznamov nefunguje vôbec a užívateľovi je tak ponúknutá normalizácia pre polia, ktoré ani nevyplnil. Odstráneniu tohto problému sa venuje kapitola 4.5

- Po uložení záznamu je užívateľ presmerovaní na editáciu záznamu, bez toho aby vedel, že uloženie sa podarilo. Užívateľ môže z takéhoto presmerovanie nadobudnúť pocit, že systém nefunguje správne. Doplneniu hlášok sa venuje kapitola 4.6.

- Pri prepise jednotlivých záznamov je možné zobraziť priamo vo formulári sken, z ktorého uživateľ prepisujeudaje. Orientácia na niektorých skenoch je zložitá vzhľadom na množstvo údajov, ktoré sa na skene nachádzajú. Preto bude do systému doplnená možnosť daný záznam ohraničiť. Bližší opis implementácie bude v kapitole 4.8.

[9] Autocomplete search je funkcia, ktorá vám pomáha rýchlo a jednoducho vyhľadať položku, ktorú hľadáte. Na základe vloženého textu as automaticky zobrazia návrhy




Obr. 3.6: Chybne nastavená responsibilita pri prepise matričných záznamov na zariadení so šírkou 1600px

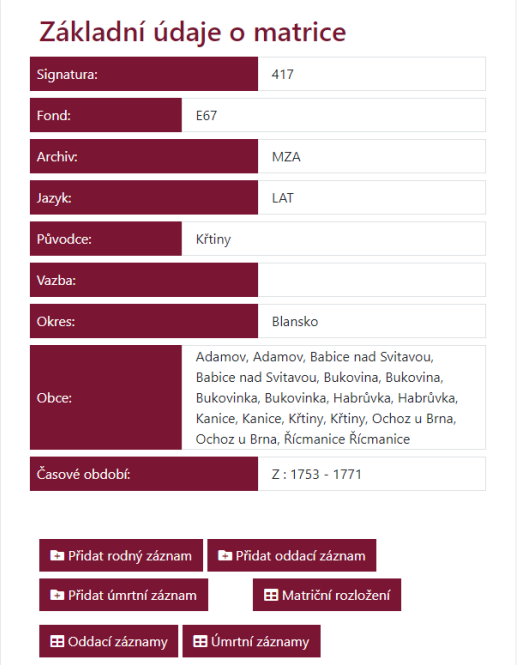
- Pri editácií ľubovolného záznamu sa v hlavičke zobrazuje popis označujúci, že ide o vytvorenie nového záznamu. Keďže sa jedná o rovnaký formulár ide o drobnú zmenu a nebude sa jej preto venovať samostatná kapitola. V projekte však boli nájdené aj iné chyby týkajúce sa duplicity kódu (4.7.1), síce sa nejedná o chyby, ktoré by ovplyvňovali samotnú funkčnosť riešenia, avšak z pohľadu užívateľa sa môžu prejaviť práve takýmito drobnosťami. Práve drobné chyby a jemné detaily užívateľského rozhrania ovplyvňujú rozhodnutie užívateľa, či aplikáciu použije aj v budúcnosti.
- Chýbajúce údaje pri niektorých záznamoch o historických prameňoch viedli k chybe na strane serveru (error 500). Zo systémových logov sme zistili, že chyba je spôsobená v šablónach, ktoré sa pokúšajú tieto chýbajúce informácie vypísať. Takáto chyba môže byť pre užívateľa mätúca, a tak ako predchádzajúce chyby môže viesť k tomu, že užívateľ stratí záujem o používanie aplikácie. Po bližšej analýze sme zistili, že táto chyba sa nevyskytuje pri matrikách, a teda rovnako ako predchádzajúci nedostatok pramení z duplicitného kódu.

### 3.4.3 Chybné dáta

V Českej republike je veľa obcí s rovnakým názvom a pri vkladaní údajov do databázy nebol tento fakt zohľadnený. Oblasti pokrytia jednotlivých záznamov historických prameňov sú chybné priradené k prvému výskytu obce v tabuľke. Táto chyba rozbíja celé usporiadanie jednotlivých záznamov. Správne data sme preto museli nahrať nanovo. Na vkladanie chýbajúcich údajov je zameraná kapitola 4.1.

### 3.4.4 Duplicitný kód

Po nahliadnutí do zdrojového kódu systému DEMoS sme objavili, že niektoré časti kódu sa opakujú viackrát v rôznych častiach aplikácie. Duplicitný kód môže mať niekoľko negatív-



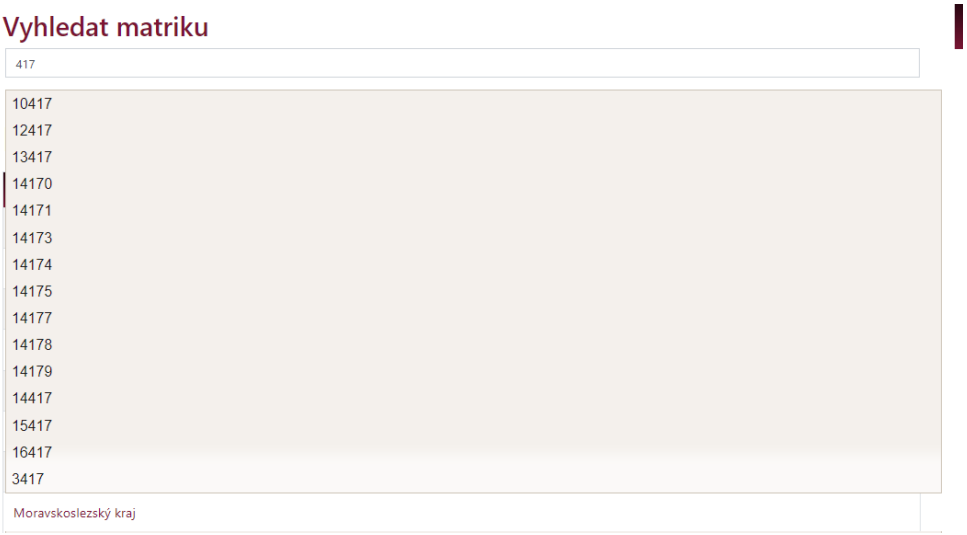

Obr. 3.7: Chybne nastavená responsibilita hlavičky s údajmi o matrike na zariadení so šírkou 600px

nych dôsledkov. Po prvé, zvyšuje množstvo kódu, ktorý musí byť udržiavaný. Ak sú úpravy potrebné v kóde, ktorý sa duplikuje, musia byť vykonané v každej jeho kópie, aby sa zabezpečila konzistencia systému. To môže zvýšiť čas a náklady na údržbu systému. Okrem toho duplicitný kód zvyšuje riziko chýb. Ak sa niektorá kópia kódu zmení, ale ostatné zostanú nezmenené, môže to viesť k nekonzistencii alebo nesprávnemu správaniu aplikácie. Takéto chyby je zväčša ťažké odhaliť a opraviť, čo môže viesť k strate dôveryhodnosti systému.

Duplicitný kód bol odhalený na 2 miestach. Prvým sú presentéry jednotlivých historických prameňoch, kde sa opakujú mnohé funkcie. Odstráneniu tejto chyby sa venuje kapitola 4.7.2. Ďalej sa duplicitný kód vyskytuje vo všetkých šablónach. Refaktoringu šablón sa venuje osobitná kapitola 4.7.1.

### 3.4.5 Chýbajúca funkcionalita

Napriek tomu, že systém na prvý pohľad vyzerá kompletne, chýba podpora pre niektoré historické pramene. Pozemkové knihy, sčítací operáty a soupisy poddaných dle víry nemajú vytvorený model ani presenter, taktiež pre ne chýbajú tabuľky v databázach. Najviac prepracované sú matriky, na ktorých sa vyvíjala všetka funkcionalita systému. Ostatné historické záznamy sú akýmsi pokusom o zkopírovanie funkcionality z matrík. Na mnohých miestach je funkcionalita zakomentovaná, alebo nie je dopracované prepojenie GUI s backendom aplikácie. Niektoré nedostatky budú odstránené spolu s duplicitným kódom. Veľmi veľa funkcionality je rovnaká pre všetky historické pramene. Jedná sa predovšetkým o radenie podľa polohy, vykresľovanie základných údajov, prácu s vyhľadávaním alebo frontendovú validáciu. Týmto problémom sa venuje kapitola 4.7. Doplneniu tabuliek do databáze sa bude venovať samostatná kapitola 4.9.




Obr. 3.8: Autocomplete search pri vyhľadávaní matrík podľa signatúry

### 3.4.6 Nevyhovujúce preklady

Systém DEMoS nemá implementovanú podporu pre preklady, avšak pri vývoji sa počítalo s tým, že táto funkcionalita bude v budúcnosti doplnená. Trieda **MyTranslator** obsahuje tabuľku českých prekladov, rozšírenie o ďalšie jazyky, tak bude zahŕňať iba doplnenie prekladov do iného jazyku a prepojenie s GUI, aby si užívateľ mohol vybrať preferovaný jazyk. Každému prekladanému slovu je priradený špeciálny kód na základe ktorého je vybraný preklad v tabuľke prekladov. Pri písaní nového kódu sme sa snažili využiť túto triedu na české preklady, avšak zistili sme, že ak náhodou zabudneme doplniť preklad do tabuľky aplikácia si s tým neporadí a klientovi vráti internal server error. Takáto logika určite nie je správna a drobná chyba v prekladoch nemôže spôsobiť pád aplikácie. Z pohľadu vývojára určite nie je prijemné pre každé slovo vytvárať jedinečný kód a následne ho pridávať do tabuľky prekladov. Jedná sa o pracnú činnosť, čo automaticky zvádza k tomu, odložiť preklady na neskôr. Okrem iného, pri ladení aplikácie, nič nehovoriace kľúčové kódy v šablóne znižujú schopnosť orientácie a predlžujú tak čas potrebný na odstránenie chyby. Ideálnym prístup by bolo, ak by namiesto kódov boli použité české alebo anglické názvy. V prípade chýbajúceho prekladu tak aplikácia môže užívateľovi ponúknuť aspoň aspoň nejaký výraz namiesto pádu aplikácie.

# Kapitola 4

# Implementačná časť

## 4.1 Nahrávanie matričných záznamov

Systém disponuje slušným množstvom nahraných matričných záznamov. Avšak je potrebné doplniť aj ostatné historické pramene. Súbežne s touto prácou sa pracuje na sťahovaní historických prameňoch priamo z archívov. Tieto údaje nám boli poskytnuté ako súbory vo formáte json. Aby sme boli schopný tieto dáta importovať do našej databáze vytvorili sme triedu **importDataCommand**, ktorý je toho schopný. Na nahrávanie dát použijeme knižnicu Console (3.1.3), ktorá nám umožní spúšťať PHP kód priamo z príkazovej riadky. Vstupné dáta pre tento skript je potrené umiestniť do adresára *www/data/new*.

Aby bolo možné skripty spúšťať vytvorili sme v adresári *bin* skript **console.php**, ktorý načíta aplikáciu a umožní spúšťanie skriptov z príkazovej riadky. Console ponúka intuitívne prostredie a spustení skriptu **console.php** ponúkne zoznam dostupných skriptov ktoré ponúka aplikácia. Nami vytvorený skript sa spústa pomocou príkazu *php ./console.php data:import*. Skript je možné volať s rôznymi prepínačmi, na analýzu dát, vkladanie nových záznamov, update už vložených záznamov a mnohé iné. K skriptu je dostupná aj nápoveda po volaní s prepínačom *-h* alebo *–help*. Nebudeme sa preto zaoberať tým, ako sa dá skript spustiť a na čo slúžia jednotlivé prepínače.

Po nainštalovaní knižnice Console, sme sa potýkali s problémom a nevedeli sme skripty spustiť. Odhalili sme chybnú konfiguráciu aplikácie v triede **RouterFactory**. Táto trieda smeruje požiadavky na správne presentéry aplikácie. V triede bol chybne registrovaný smerovač pre štandardizáciu predpisovaných údajov. Registráciu sme úspešne opravili.

Celý skript je navrhnutý tak, aby zachoval dáta, ktoré systém už obsahuje. Skript tak dôsledne porovnáva poskytnuté dáta s dátami v databáze. Jediný prípad, kedy sme museli pristúpiť k odstráneniu pôvodných dát je pri priraďovaní obcí k jednotlivým záznamom. V Českej republike je množstvo obcí s rovnakým názvom a dáta ktoré systém o obciach uchováva sú chybné a nemá zmysel sa snažiť ich opraviť. Preto sme tieto dáta do systému vložili nanovo.

Po analýze poskytnutých dát, sme museli mapovať niektoré údaje. Za týmto účelom sme vytvorili konštanty, ktoré su definované v triede **importDataCommand**. Jedná sa o mapovanie následujúcich údajov:

- **typy registrov** - v poskytnutých dátach boli na označenie historických prameňov použité iné označenia ako sú tie s ktorými pracuje systém DEMoS
- **jazyky** - znovu bolo potrebné mapovať označenie jazykov, tak, aby s nimi vedel pracovať systém DEMoS. Okrem toho bolo potrebné do systému niektoré jazyky doplniť.

- **označenie časového rozsahu** - rovnako bolo potrebné mapovať aj jednotlivé časové obdobia
- **typ administračnej jednotky** - mnohé typy administračných jednotiek boli mali v poskytnutých záznamoch rôzne značenie. Museli sme preto mnohé premapovať a niektoré aj doplniť do systému.

Po spustení skriptu sú načítané všetky súbory json z adresára *www/data/new*. Následné sa skript pokúša zistiť, či je daný záznam už uložený v databáze. Táto kontrola spočíva na hľadaní záznamu s rovnakou signatúrou. Následne sa porovnávajú údaje z poskytnutých dát a dát z databáze. Porovná sa archív, inventarizačné číslo, počet strán, časové obdobie, typ záznamu, jazyky. Aby bol záznam zhodný je potrebné nájsť v týchto údajoch aspoň 3 zhody. Samozrejme stačí jediná odlišnosť, aby skript vyhodnotil, že nejde o rovnaké záznamy. Následne sa podľa zvolených prepínačov volajú metódy, ktoré daný záznam ďalej spracovávajú.

Metóda **updateRecord** doplní chýbajúce údaje v databázy. Metóda **insertRecord** údaje do databázy vloží. Metóda **sortMunicips** detekuje ku ktorým v obciam sa záznam viaže. Na detekciu je využívaná štruktúra RUIÁN. Skript sa snaží na základe archívu pod ktorý daný záznam patrí nájsť správnu obec a túto závislosť medzi záznamom a obcou uloží do databázy. Metóda **getAdminUnit** slúži na uloženie archivačnej jednotky pre daný záznam. Ďalej trieda obsahuje pomocné metódy na prácu s dátami, ukladanie alebo načítanie rôznych závislosti. Po skončení skriptu sú jednotlivé záznamy rozdelené a uložené do priečinkov *www/data/crashed*, *www/data/paired*, *www/data/inserted* podľa toho ako dopadla operácia nad týmito záznamami. V prípade že nastala chyba sú ku každému záznamu doplnené chybové hlášky, v prípade, že záznam bol spárovaný so záznamov v databáze, je doplnené id pod ktorým záznam figuruje v systéme.

Počas vypracovávania práce nám boli poskytnuté aj údaje o pozícii jednotlivých záznamov vrámci skenu, bohužiaľ tieto dáta neobsahovali údaje o rozložení pôvodného skenu. Nakoľko jednotlivé skeny môžu obsahovať buď samotnú stránku alebo dvojstránku systém potrebuje vedieť či sa sken nachádza na ľavej alebo pravej strane, prípadne cez celý sken. Nakoľko systém nie je schopný bez tohto údaju pracovať neboli tieto dáta ani importované.

## 4.2 Upravenie tabuliek s prepísanými záznamami

Pridaním vlastnosti sticky do záhlavia tabuľky naráža na limity tejto vlastnosti. Vlastnosť sticky nefunguje pokiaľ jeden z jej rodičovských elementov upravuje pretečenie (vlastnosť overflow). Nakoľko sa jedná o širokú tabuľku je obalená elemente s triedou *table-responsive*, ktorá danému elementu nastaví šírku na 100% a povolí pretečenie na x-ovej osi. Ako jednoduché riešenie sa naskytuje presunúť vlasnosť pretečenia priamo na jednotlivé riadky tabuľky, a vlastnosť sticky nechať v nadradenom elemente záhlavia tabuľky. Keďže tabuľky v HTML fungujú ako celok nie je možné vlastnosť pretečenia vložiť dovnútra tabuľky.

V článku od Dannie Vinthera[26] je uvedených niekoľko riešení ako daný problém vyriešiť. Jedným z nich je elementu s vlastnosťou pretečenia nastaviť pevnú výšku, toto nastavenie odstráni negatívny vplyv pretečenia na vlastnosť sticky. Druhým riešením ktoré Dannie v článku opisuje, je oddelenie záhlavia tabuľky od obsahu. Následne záhlavie tabuľku obaliť do elementu s vlastnosťou pretečenia, a to následne obaliť do elementu s vlastnosťou sticky. V našom prípade však tabuľke nemôžeme nastaviť pevnú výšku a preto sa pozrieme na riešenie s oddelením záhlavia. Dannie v článku nepracuje s tabuľkami ale s obyčajnými div elementami, s pevnou šírkou. Keďže v našej tabuľke potrebujeme mať istú variabilitu medzi

šírkou jednotlivých stĺpcov museli by sme použiť rozloženie grid[1]. V našom prípade by sme tak použili jeden grid pre záhlavie a druhý pre obsah tabuľky. Problém však nastáva v bode, kedy si uvedomíme, že potrebujeme zosynchronizovať šírku jednotlivých stĺpcov medzi záhlavím a obsahom. Rozloženie grid ponúka možnosť automatickej šírky pre riadky a stĺpce rozloženia, čo by sme samozrejme museli využiť, keďže potrebujeme zachovať variabilnú šírku stĺpca. V takom prípade nie sme schopní stĺpcom nastaviť šírku, ktorú potrebujú.

Je jasné, že sa jedná o problém, ktorý nie je možné vyriešiť jednoduchým pridaním pár vlastnosti. Z predchádzajúcich možností vieme, že potrebujeme oddeliť, záhlavie tabuľky od obsahu, ale zároveň potrebujeme zachovať tabuľku, ktorá neumožnime toto oddelenie. Ako priamočiare riešenie sa naskytlo skopírovať záhlavie tabuľky a vložiť ho ako novú tabuľku pred tú súčasnú. Správnym nastavením nového záhlavia, vieme dosiahnuť, že jednotlivé záhlavia tabuliek budú na rovnakej pozícii. Skopírované záhlavie tak môžeme v správnom poradí obaliť do elementov a nastaviť im požadované vlastnosti. Nakoľko tabuľky obsahujú veľké množstvo údajov je nezmyselne vytvárať v šablóne dve zhodné tabuľky, obsah záhlavia radšej skopírujeme pomocou metódy **copyHeader**. O nastavenie rovnakej šírky pre jednotlivé stĺpce sa postará metóda **syncWidth**. Pri tvorbe metódy **syncWidth** bolo potrebné dbať na fakt, že jednotlivé bunky tabuľky v sebe obsahujú ďalšie tabuľky, a teda treba synchronizovať iba stĺpce tej vonkajšej. Synchronizáciu šírky nám značne uľahčilo to, že pôvodná tabuľka obsahuje svoje záhlavie, a teda nebolo nutné kontrolovať, či šírka daného stĺpca v záhlaví nie je väčšia ako šírka toho istého stĺpca v obsahu. Obe tieto metódy sú obsiahnuté v súbore *sticky-table.js* a sú volané hneď po načítaní stránky. Synchronizácia šírky je taktiež volaná pri každej zmene šírky okna. Aby pri načítaní stránky užívateľ nemal pocit, že je tabuľka rozbitá, nové záhlavie bude schované a na stránke sa zobrazí až po nastavení šírky. Aby sme synchronizovali scrollovanie záhlavia spolu so scrollovaním obsahu tabuľky použijeme skript, ktorý opisuje Dannie vo svojom článku. Elementom, ktoré potrebuje synchronizovať stačí nastaviť triedu *syncscroll* a nastaviť im rovnaký atribút name. Veľmi podobným spôsobom pridáme k tabuľke aj scrollbar s vlastnosťou sticky.

Aby sme uľahčili pridanie tejto funkcionality k ľubovolnej tabuľke a nevytvárali zbytočne duplicitný kód, ktorému sa venuje kapitola 4.7.1, vytvoríme 2 šablóny s kostrou záhlavia a kostrou scrollbaru. Tieto šablóny sú uložené v súboroch *stickyTableHeader.latte* a *stickyTableScrollber.latte*. Nastavenie tabuľku tak spočíva v includovaní týchto šablón a pridaním tried *sticky-table-body-wraper* a *syncscroll* rodičovskému elementu pôvodnej tabuľky, ďalej pridaním atribútu *name=myElements* tomu istému prvku a pridaním atribútu *id=sticky-table-body* pôvodnej tabuľke. Pridanie tried a atribútov je potrebné pre správne prekrytie záhlaví a taktiež pre správne fungovanie javaScriptu, ktorý nastaví šírku nového záhlavia.

Ako ďalšie uľahčenie orientácie v tabuľkách sme doplnili možnosť skryť jednotlivé stĺpce tabuliek. Nastavenie tejto možnosti je možné po otvorení okna nástrojov. Okno nástrojov sme rovnako ako pri prepise záznamov schovali do plávajúceho tlačítka v pravej spodnej časti okna, aby bolo zachované rovnaké používanie nástrojov na celej stránke a nemiatli sme užívateľa. Funkcionalitu zaisťuje niekoľko javaSkriptových metód definovaných v súbore *table-column-shower.js*. Metóda **fillModal**, ktorá je volaná po načítaní stránky skopíruje jednotlivé názvy stĺpcov a vloží ich do modálu[2] spolu s checkboxom, ktorý reprezentuje, či je daný stĺpec zobrazený alebo schovaný. Metóda okrem iného zistí, či sa v danom stĺpci

[1] CSS grid umožňuje tvorbu mriežkových rozložení na webových stránkach. Je to technika, ktorá umožňuje presné rozloženie elementov na stránke v riadkoch a stĺpcoch.

[2] Modal je interaktívny prvok používaný v webovom dizajne na zobrazenie obsahu, ktorý sa objaví vo vyskakovacom okne nad aktuálnym obsahom stránky.

nachádza aspoň jedna bunka s obsahom a túto informáciu uloží ako atribút checkboxu. Metóda **handleClick** reaguje na zmenu checkboxu a schová alebo zobrazí daný stĺpec. Jednotlivé stĺpce maju priradené id, ktoré reprezentuje ich poradie. To či sa daná bunka v tabuľke zobrazí je vypočítane na základe tohto id a poradia bunky za využitia matematickej operácie modulo. Metóda **showEmpty** na základe atributov checkboxov, zobrazí alebo schová stĺpce, v ktorých nie sú žiadne údaje. Metóda **showAll** zobrazí všetky stĺpce, rovnako ako metóda **showEmty** reaguje na stlačenie tlačidla.

## 4.3 Responsibilita webovej aplikácie

Stránky na vkladanie nových záznamov majú nesprávne nastavenú responsibilitu. Kombinujú sa tu triedy z knižnice bootstrap (3.1.5) spolu s vlastnými štýlmi a javaScriptom. Použitie bočného navigačného panela nie je úplne štandardné riešenie, a preto použitie bootstrapu treba doplniť o vlastné css štýly. Nie je však potrebné použitie vlastných javaScriptov. Prvý konflikt nastáva, použitím metódy **checkHelperVisibility**, ktorá sa stará o schovanie navigačného panela na základe šírky okna. Súčasne s touto metódou panel obsahuje triedu *d-xl-block*, ktorá panel zobrazí iba ak je šírka okna väčšia ako bootstrapom preddefinovaný break point xl (1200px). Táto metóda je teda zbytočná, preto ju odstránime. Celá stránka je obalená v elemente s triedou *container-with-helper*, samotný panel je označený triedou *helper*. Obe tieto triedy majú vytvorené štýly podľa šírky okna. Tu je jednoznačným problémom nastavenie break pointov pre zmenu štýlov, ktoré nie je kompatibilné s bootstrapom. a taktiež zarovnanie obsahu stránky na pravú stranu, ktoré pri väčších rozlíšeniach spôsobuje netypické zarovnanie stránky. Nové riešenie ša odráža od maximálnej šírky obsahu stránky nastavené bootstrapom. Bootstrap obsah zarovná do stredu stránky a nastaví mu automatické okraje. My následne upravíme veľkosť ľavého okraja tak, aby bola vždy väčšia ako šírka navigačného panelu. Nakoľko css nevie nastaviť minimálnu hodnotu jedného okraja, za pomoci css funkcie **max** priradíme ľavému okraju väčšiu hodnotu. Nakoľko nepoznáme veľkosť okraja, musíme si ju vypočítať pomocou css funkcie **calc**, kde od šírky okna odčítame veľkosť obsahu a túto hodnotu vydelíme 2, druhá hodnota je šírka panela s drobnou rezervou. Takýmto nastavením dosiahneme prirodzeného zarovnania na stránke, a vieme, že bočný panel nikdy neprekryje obsah stránky.

Pri tvorbe hlavičky s údajmi o historickom prameni sa nikto nezamýšľal nad responsibilitou a šírky jednotlivých hlavičiek boli nastavené na pevno. Bootstrap používa na rozmiestnenie na stránke grid s 12 stĺpcami. Nakoľko sú údaje rozdelené do 3 stĺpcov a stredný obsahuje najviac informácii je logicky najširší. Pri použití mobilného zariadenie sa však všetky údaje zoradia do jedného stĺpca a vznikajú tak nezarovnané hlavičky, ako je znázornené na obrázku 3.7. Toto zarovnanie sme upravili jednoduchým pridaním tried, ktoré pre nízkych rozlíšeniach použijú iné rozloženie v gride. Nepravidelné zarovnanie tlačidiel pod hlavičkou, ktoré zhodne ako hlavička používa uchytenie do mriežky pomocou gridu, zmeníme na umiestnenie pomocou vlastnosti float float. Táto vlastnosť nám umožní nastaviť pravidelné medzery medzi jednotlivými tlačidlami a pri zmene šírky, ich jednoducho preusporiadame pod seba.

## 4.4 Úprava vyhľadávania archívnych jednotiek podľa signatúry

Vyhľadávanie archívnych jednotiek je sa nachádza na stránke s výberom kraja. Jedná sa o druhý krok pri výbere územnej hierarchie pre daný historický prameň. Nakoľko si užívateľ už vybral historický prameň, museli sme to zohľadniť pri filtrovaní výsledkov. Do metódy **createSearchForm**, ktorá vytvára samotný formulár na vyhľadávanie sme pridali skrytý formulárový prvok s názvom *registry_search_type*. Formulár je automaticky odosielaný po každej zmene ľubovolného prvku. Aby sme počas vkladania hľadanej signatúry zabránili odosielaniu veľkého množstva požiadavkov na server, zvýšili sme čas kedy funkcia čaká na prípadný ďalší vstup od poslednej zmeny na 500 ms z pôvodných 100ms.

O obslúženie tohto ajaxového požiadavku sa stará metóda **getRegisterBySignature** v triede **AutocompletePresenter**. Metódu sme museli rozšíriť vstupný parameter reprezentujúci typ hľadaného historického prameňa. Východziu hodnotou tohto typu je *null*, aby bolo možné hľadať vo všetkých historických prameňoch. Daná metóda vyhľadáva v databáze za pomoci triedy **registerManager**. Vyhľadávanie sme upravili tak, že hľadáme historické pramene s presnou zhodou v signatúre a taktiež hľadáme 15 výsledkov s čiastočnou zhodou. Výsledky týchto dvoch hľadaní sa reťazia, tak, že ako prvá možnosť sa vracia presná zhoda a následné sa radia abecedne všetky ostatné výsledky. Vytvorli sme metódu **createLabelSignatureSearch**, ktorá pre daný výsledok zostaví štítok, opisujúci daný výsledok. Štítok obsahuje informácie v následujúcom poradí: signatúra, skratka archívu, kraj, okres a obec ktorá daňy historický prameň spravuje.

## 4.5 Opravenie GUI pre normalizáciu v prepisoch záznamov

Užívateľ má možnosť pri prepise matričných záznamov využiť normalizáciu všetkých názvov a mien, ktoré pri prepise zadal. GUI pre normalizáciu môže v určitých prípadoch dosiahnuť aj viac ako 300 formulárových prvkov. K tomuto vysokému číslu prispieva fakt, že jednotlivé osoby môžu mať až 10 mien a taktiež to, že systém pri prepisa ponúka možnosť prepísať veľké množstvo osôb. Preto dáva preto zmysel, aby sa možnosť normalizácie zobrazovala iba pre prepísané údaje. Toto zobrazovanie jednotlivých prvkov normalizačnej časti však bolo implementované iba pri prepise rodných záznamov, avšak nefungovalo 100%. Pri prepise dodacích u úmrtných záznamov nefungovalo vôbec. Formulár na prepis údajov je rozdelený do sekcii, pričom každá sekcia sa zaoberá jednou osobou. Pri všetkých osobách sa prepisujú podobné informácie, ako napríklad meno, priezvisko, titul, dátum alebo miesto narodenia.

Aby sme uľahčili vytváranie normalizačného formulára a predišlo tak rôznym chybám, vytvorili sme systém, ktorý na základe zoznamu osôb a zoznamu údajov vytvorí normalizačnú časť formulára. Pri vytváraní formuláru definujem zoznam *norm_fields*, kde sú ako kľúče použité sekcie. Ku každej sekcii - osobe, je ďalej definovaný český názov, názvy polí danej sekcie, ktoré sa budú normalizovať a počet osôb ktoré je takto možne normalizovať. Následne v metóde **initNormFields** sú na základe počtu opakovaní osôb doplnený pôvodný zoznam tak, aby každá osoba predstavovala jeden záznam v zozname. Na základe takto vytvoreného zoznamu sa zostaví normalizačná časť formulára.

JavaScript *normalizationAddEditRecord.js*, ktorý pracuje s normalizáciou máme k dispozícii, a na jeho použitie je potrebná správna inicializácia. Na inicializáciu je potrebné zadať rovnaké údaje o sekciách, a normalizovaných poliach v sekcii rovnako ako v nami vytvorenom zozname. Na základe tohto zoznamu je možné pohodlne a bez chyby správne nastaviť

inicializáciu. Aby normalizácia fungovala správne bolo potrebné v javaScripte upraviť názov pre krstného otca a pre svedka, tak ako je použitý inde v systéme.

## 4.6 Doplnenie hlášok o výsledku operácie

Hlášky o výsledku operácie sú neoddeliteľnou súčasťou systému, na ich vytvorenie sme použili **flashMessage**, jedná sa o interaktívnu komponentu Nette aplikácie. Takto vytvorené správy sú uložené do session[3] a sú tak dostupné aj po presmerovaní užívateľa na inú stránku. Do systému sme doplnili hlášky o uložení záznamu, a ku presmerovaniu z editácia záznamu, ak užívateľ nemá oprávnenie na editáciu. Po uložení záznamu do DB, dochádzalo k presmerovaniu na editácia záznamu. Toto presmerovanie bolo volané zo zlými parametrami, kde dochádzalo k strate správ uložených v session. Aby sme o uložené správy neprichádzali, museli sme toto presmerovanie upraviť. Chyba sa vyskytla pri ukladaní prepisov všetkých historických prameňov.

## 4.7 Odstránenie duplicitného kódu

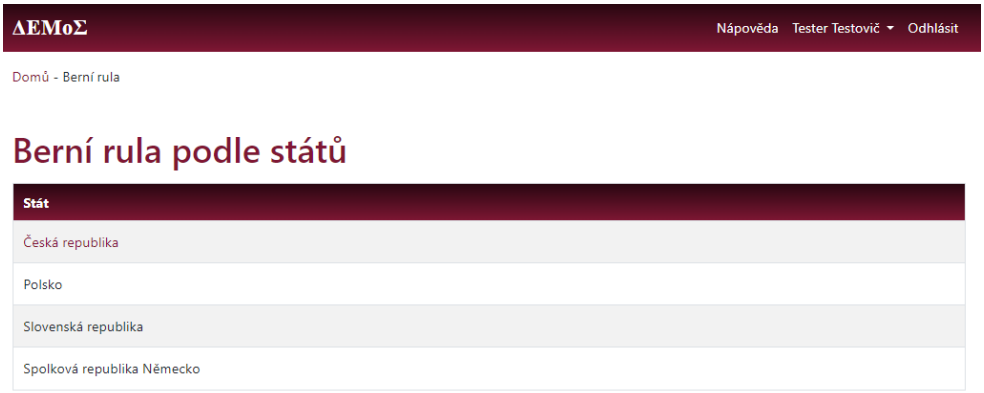
Táto kapitola sa venuje odstráneniu duplicitného kódu. O tom, že duplicitný kód je pri vývoji nežiaduci a o jeho následkoch píšeme v kapitole 3.4.4. Ako sme už spomínali, aplikácia obsahuje 8 historických prameňov a každý má svoj vlastný presentér aj šablóny, ktoré ho vykresľujú. Názvy jednotlivých presentérov vychládajú z anglických názvov pre jednotlivé historické pramene. Tieto názvy sú uvedené v kapitole 3.3. Treba podmoknúť, že z 8 historických prameňov chýba implementácie pre pozemkové knihy, sčítací operáty a soupis poddaných dle víry. Úpravami, ktoré urobíme, doplníme základnú funkcionalitu aj pre tieto chýbajúce pramene. Pramene sú na stránke radené hierarchicky podľa oblasti. Toto rozdelenie je pre všetky pramene rovnaké a preto dáva zmysel, aby aj funkcionalita bola jedna. Podrobne sa na túto funkcionalitu pozrieme v nasledujúcich kapitolách.

### 4.7.1 Použitie blokov v šablónach Latte

Vytvoríme adresár *templates/common*, ktorý obsahuje šablóny, ktoré sú spoločné pre všetky presentery. Ako prvé sme sa pozreli na stránku kde užívateľ vyberá krajinu. Vytvorili sme šablónu *commonState.latte*, do ktorej sme premiestnili pôvodný obsah týchto stránok. Jediný rozdiel, ktorý je medzi pôvodnými šablónami je názov hlavičky stránky a navigácia, ktorá užívateľovi ukazuje, v ktorej časti stránky sa nachádza. Ako táto stránka vyzerá je znázornené na obrázku 4.1. Pretože s názvom hlavičky budeme pracovať aj na iných stránkach budeme ho vkladať do každej šablóny. Aby bolo zaručené, že názov hlavičku bude v každej šablóne daného presentéru, budeme ho vkladať ako *sectionName* v metóde, ktorá sa volá pred generovaním každej stránky - **beforeRender**.

pre navigáciu tiež vytvoríme samostatnú šablónu *commonNavigation.late* s blokom *navigation*. Navigácia sa skladá z názvu historického prameňa, hierarchie oblasti, od štátu až po okres, archívu, fondu, signatúry daného historického prameňa a na poslednej úrovni je akcia, ktorá sa odkazuje na zoznam prepísaných údajov daného prameňa. Aby sme mohli pre všetky stránky použiť rovnakú navigáciu, využijeme funkciu **ifset**, ktorú nám ponúka šablónovací systém Latte. Takýmto spôsobom zaručíme, že sa vykreslí iba tá časť navigá-

[3]Session je dočasné úložisko informácií, ktoré sú priradené k jedinečnému identifikátoru a uchovávané počas určitého časového obdobia, kým trvá interakcia medzi klientom a serverom.




Obr. 4.1: Stránka na výber štátu

cie, ktorá bola vložená do šablóny. Nakoľko každá časť navigácie je klikateľným odkazom, ktorý obsahuje link na rovnaký presenter, využijeme, že ak pri tvorbe odkazov v systéme Latte nie je vyplnený presenter, šablóna doplní ten stávajúci. Pre všetky presentery sú akcie rovnaké až na poslednú úroveň. Názov tejto akcie vložíme do šablóny rovnako ako názov sekcie pod názvom *recordAction*. Na stránke so zoznamom prepísaných záznamov nechceme aby bola posledná úroveň odkazom, nakoľko by odkazovala na aktuálnu stránku. Preto si v bloku definujeme pomocou latte makra **define** východziu hodnotu *recordActive* na hodnotu false. Na základe tejto hodnoty následné vložíme do stránky odkaz alebo čistý text. Ak budeme chcieť aby bola posledná úroveň odkazom, jednoducho pri inklúdovaní bloku do inej šablóny, definujeme túto hodnotu na true.

Rovnako ako pri stranke s výberom štátov budeme postupovať aj pri stránkach pre výber kraja - *commonState.latte*, výber okresu - *commonDistrict.latte* a pre výber obce *commonMunicipality.latte*. Nette automaticky vykreslí šablónu, ktorá je pomenovaná rovnako a je uložená v súbore presentéru. Aby sme v projekte nemali také množstvo šablón, ktoré bude Nette vykresľovať na základe tohto nastavenia, nastavíme v metódach, ktoré obsluhujú danú akciu, že sa majú vykresliť nami vytvorené šablóny. Použijeme na to metódu **setFile**.

Pri stránke so zoznamom prepísaných údajov je rovnaká navigácia, a hlavička s údajmi o prameni. Navigáciu už máme vytvorenú, pre spoločnú hlavičku vytvoríme šablónu *commonRegisterHeader.latte*, údaje sú tam zakaždým rovnaké, takže postup bude rovnaký ako v predchádzajúcich prípadoch. Pri matrikách sú zoznamy 3, konkrétne pre prepis, narodí, sobášov a úmrtí. Stránka obsahuje tlačidlá, ktoré umožňujú prepínanie medzi týmito typmi záznamov. Celú vrchnú časť stránky vytvoríme v šablóne *registerHeader.latte*. To čo sa bude meniť, sú samotné tabuľky, pretože tabuľky sú obšírne, vytvoríme pre na samostatné šablóny, aby sa uľahčila orientácia v kóde. Celú stránku potom vykladáme z jednotlivých šablón. Stránka pre rodné matriky bude vyzerať následovne:

```
{block content}
    {include "registerHeader.latte"}
    {if count($records) > 0}
        {include 'birthRecordTable.latte'}
    {/if}

{/block}
```

```
{block customScripts}
    <script src="{$basePath}/js/syncscroll.js"></script>
    <script src="{$basePath}/js/sticky-table.js"></script>
    <script src="{$basePath}/js/table-column-shower.js"></script>
{/block}
```

Pre stránky na prepis záznamov, budeme potrebovať vytvoriť viacero šablón, do ktorých uložíme duplicitný kód. V šablóne *commonScan.latte* vytvoríme blok *scanContainer*, ktorý bude obsahovať časť stránky, ktorá zobrazuje prepisované skeny. Vytvoríme aj druhý blok *scanScripts*, ktorý bude obsahovať všetky potrebné skripty pre fungovanie zobrazovania skenov, ako aj inicializáciu triedy **imageZoom**. V šablóne *commonTooltip.latte* bude nápoveda, k používaniu prepisovacieho rozhrania. V šablóne *commonTools.latte* vytvoríme blok *toolsButton* pre tlačítko s nástrojmi. V bloku *toolsModal* budú uložené všetky modály ktoré sa používajú pri prepisoch. Tu bolo potrebné jednotlivé modaly premenovať, pretože ich mená sa pôvodne odvíjali od názvu historického prameňa, ktorý sa na stránke prepisuje, ale ich obsah bol rovnaký. Do bloku *toolsScripts* sme vložili všetky skripty, ktoré plnia jednotlivé modály obsahom alebo zabezpečujú ich funkcionalitu. Tento blok je potrebné inklúdovať s parametrom *formName*, aby skripty spracovali s formulárom, ktorý je na stránke. Samotný formulár sa vkladá pomocou makra **control**. Aby sme mohli makro v šablóne použiť, musí byť v presentéry definovaná metóda, ktorá tento formulár zostaví. Po vytvorení týchto šablón ostala v pôvodnej šablóne iba štruktúra stránky, nastavenie frontendovej validácie a inicializácie normalizácie.

### 4.7.2 Vytvorenie spoločného presentéru pre historické pramene

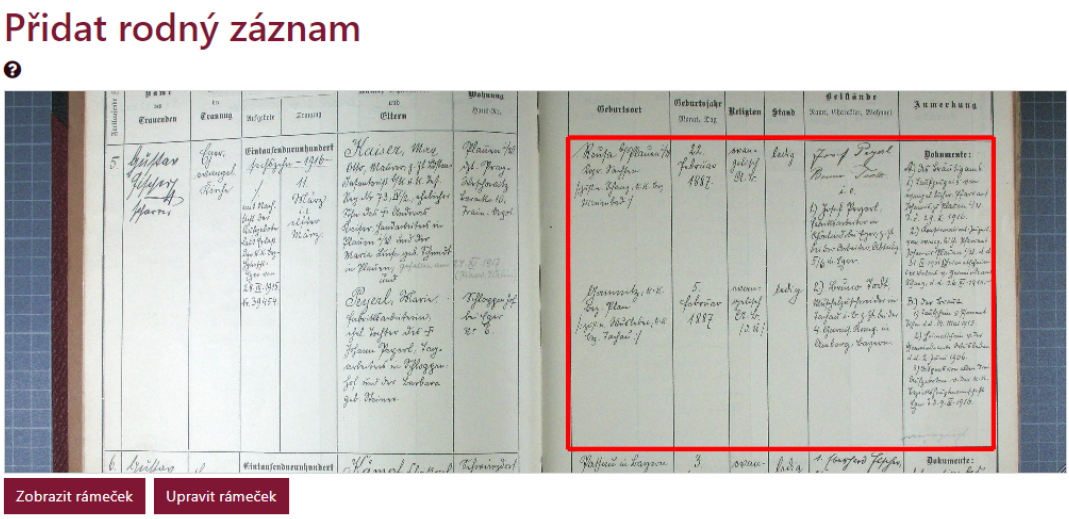
Aby sme mohli odstrániť duplicitný kód z presentérov pre historické pramene, vytvori sme rodičovský presentér **RecordPresenter**, z ktorého sa bude daná funkcionalita dediť. Presentéry už majú rodičovský presentér **BasePresenter**, ten sme však nemohli pre tieto účely použiť nakoľko z neho dedia aj iné presentéry ako tie, ktoré upravujeme. Do nášho presentéra sme doplnili metódu **renderDefault**, ktorá v niektorých presentéroch chýbala. Táto metóda klienta presmeruje na stránku s výberom krajín, ktorá je ako prvá v územnej hierarchii. Tento presentér obsahuje všetky metódy pre vykresľovanie územnej štruktúry. Jedná sa o tieto metódy: **renderStates**, **renderState**, **renderRegion**, **renderDistrict**. Zároveň v týchto metódach nastavíme východziu šablónu, ktorá ich bude vykresľovať. Na stránke s výberom obce sa zobrazujú jednotlivé historické pramene. Tieto pramene sú z databázy získane pomocou metódy **getByTypeForMunicipality** definovanej v triede **RegisteManager**. Pri volaní tejto metodý je potrebné vedieť, o aké typy historických záznamov máme záujem, preto sme vytvorili metódu **renderMunicipalityByType**, ktorá obsahuje všetku funkcionalitu. Bude volaná z metódy **renderMunicipality**, s rovnakymi parametramy a typom historického prameňa. Aby sa uľahčilo používanie typov, v každom presentéry je definovaná konštanta *REGISTRY_TYPE*, a v novom presentéry *ALL_TYPES*. Konštanta *ALL_TYPES*, je zároveň aj preddefinovanou hodnotou argumentu *type* metódy **renderMunicipalityByType**. Rovnako sme vytvorili metódu **createSearchForm**, ktorá je volaná s metódy **createComponentSearchForm**, spolu s typom historického prameňa. Do pôvodného rodičovského presentéru sme pridali metódu **setupRegisterAdminUnit**, ktorá z id matričného prameňa vyhľadá údaje o administračnej jednotke daného prameňa a vloží ich do šablóny. Táto metóda sa používa vo všetkých vyššie spomínaných metódach, ale aj v iných presentéroch ako pre historické pramene.

Metóda **handleCheckExistingRecord** slúži na obsluhu ajaxového požiadavku a zisťuje či daný záznam už nebol v minulosti prepísaný. Táto metóda bola vytvorená iba pri prepise matričných rodných záznamov. My sme metódu rozšírili, tak aby bola univerzálna. k pôvodným parametrom *registerId*, *scan*, *pos* a *lay*, sme pridali parameter *formName*. Podľa pridaného parametra sa vyberá manažér, ktorý bude pracovať so správnou tabuľkou v databáze. Ďalšie úpravy boli potrebné v javascriptovej časti, kde sme v súbore *funcions.js* vytvorili metódu **bindCheckExistingRecord**, ktorá na základe názvu formulára zostaví názvy formulárových polí. Pri zmene týchto polí odošle ajaxovú požiadavku pomocou pôvodnej metódy **makeAjaxCallWithoutSnippets**.

Nakoľko v presentéroch vkladáme do šablón názvy, vytvorili sme metódu **translate**, ktorá sa stará o preklady v presentéroch. Využíva k tomu triedu **MyTranslator**. Pri preklade sa odchytáva chyba **UnexpectedValueException**, ktorá je vyvolaná chýbajúcim prekladom. Chyba je pomocou Tracy debbugeru (3.1.3) uložená do logovacieho súboru. Túto funkcionalitu sme zvolili z toho dôvodu, že chýbajúci preklad nemôže spôsobiť pád aplikácie, a užívateľovi tak poskytneme aspoň nepreloženú variantu názvu.

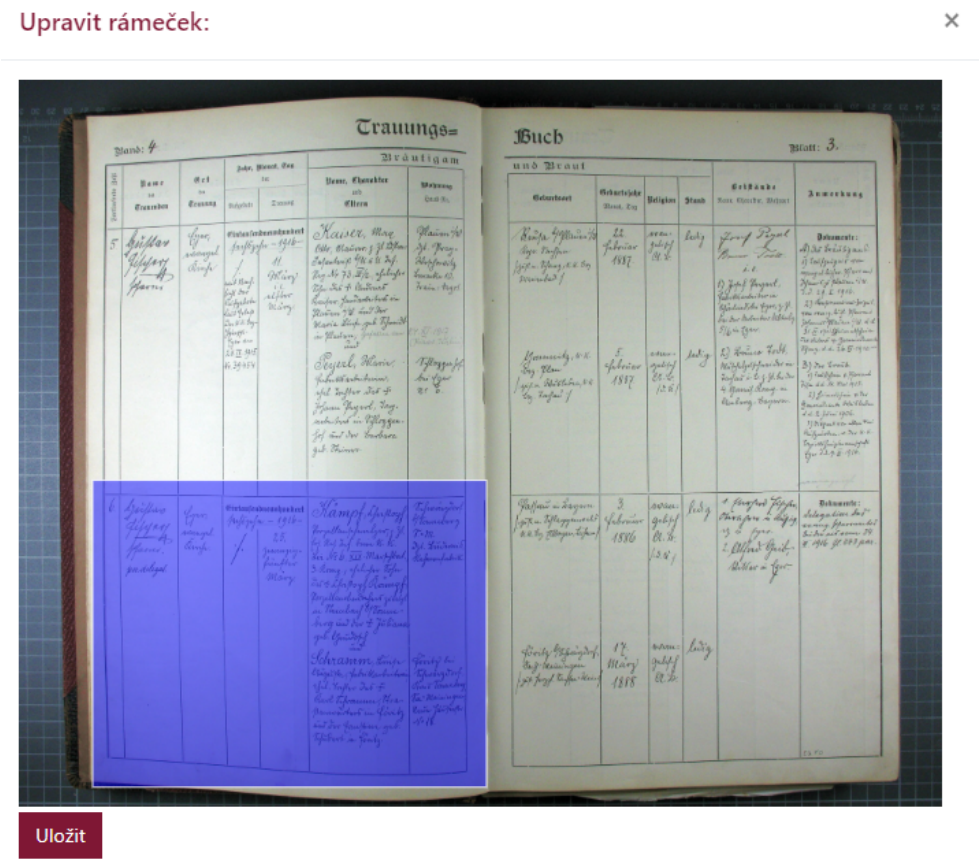
## 4.8 Orámovanie predpisovaného záznamu v skene

O zobrazovanie skenov pri prepise sa stará trieda **imageZoom** definovaná v súbore *zImgZoom.js*. Tuto triedu vytvoril Tomáš Zatloukal v svojej diplomovej práci.[27] Trieda sa stará o načítanie, zobrazenie a zoomovanie daného skenu. Ako sme už spomínali, skeny niektorých historických prameňov sú dostupné online, viac o tom, ako sú skeny uložené v databáze je popísane v kapitole 3.3.1, okrem iného databáza obsahuje aj pozície jednotlivých záznamov na skene. Nakoľko pri vytváraní danej triedy ešte neboli dostupné pozície záznamov, musíme vytvoriť tabuľku aj pre tieto údaje.




Obr. 4.2: Orámovanie predpisovaného záznamu na skene

Ako prvé sa zameriame na zobrazovaciu časť, kde bolo potrebné vyriešiť hneď niekoľko problémov. Samotné skeny sú načítané až v momente, keď sa ich užívateľ rozhodne zobraziť aby sme boli schopný zobraziť rámček k danému záznamu, musí užívateľ najskôr vyplniť ktorý záznam sa hodlá prepisovať. Na komunikáciu so serverom sme vytvorili triedu **ScanP-**




Obr. 4.3: Úprave orámovania predpisovaného záznamu na skene

**resenter**. O samotné načítanie sa stará metóda **actionGenerateBorder**, ktorá pomocou ajaxového požiadavku načíta daný rámček. Nakoľko upravovať sken priamo na strane klienta pomocou javaScriptu by bolo náročné, rozhodli sme sa, že rámček budeme generovať serverom ako obrázok s priehľadným pozadím a jednoducho ho zobrazíme pred načítaným skenom. Zobrazenie rámčeka je zobrazené na obrázku 4.2.

Rámčeky sú generované pomocou umelej inteligencie a môže sa stať, že nebudú presne zarovnané s daným záznamom, alebo môžu zakrývať nejaké údaje, doplnili sme možnosť upraviť daný rámček. Úpravu rámčeka sme riešili pomocou knižnice jQuery (3.1.4). Nakoľko súbory svg (Scalable Vector Graphics) je možné vložiť do stránky pomocou DOM (Document Object Model) elementov, jQuery tak môže pracovať s jednotlivými časťami svg. V súbore *scan-border-edit.js* sú implementované metódy **startDrag**, **moveDrag** a **stopDrag**, ktoré vyhodnocujú vstupné údaje z myši a riadia tak nakreslenie nového obrázka. Súradnice sú počas pohybu myši metódou **redraw** vkladané do svg elementu. Vytvorenie nového orámovanie je znázornené na obrázku 4.3. Na strane servera sa o uloženie nového rámčeka sa stará metóda **actionSaveNewBorder**.

## 4.9 Doplnenie chýbajúcich tabuliek do databázy

V databáze systému DEMoS sa momentálne nenachádzajú všetky potrebné tabuľku pre správne fungovanie systému. Bude potrebné vytvoriť štruktúry tabuliek pre soupis poddaných dle víry a pre pozemkové knihy. Pri vytváraní tabuliek budeme vychádzať z už vytvorených tabuliek a z údajov o týchto historických prameňoch, ktoré nám boli poskytnuté.

### 4.9.1 Soupis poddaných dle víry

Samotný súpis poddaných vznikal počas roku 1651. Predstavoval jedno z rekatolizačných opatrení, ktoré malo zmapovať náboženské rozvrstvenie českého obyvateľstva. Predchádzal mu patent českých miestodržiteľov, ktorý vrchnostiam jednotlivých panstiev nariaďoval zostaviť zoznam všetkých poddaných s určením, akého sú vyznania. Tento zoznam mali do šiestich týždňov odovzdať miestodržiteľskej kancelárii v Prahe. K patentu bol priložený formulár, ktorým sa zaisťovala jednotná forma zostavovania súpisu na celom území Čiech a Moravy.[10]

V databáze bolo potrebné pre prepis záznamov zo soupisů poddaných dle víry vytvoriť 3 tabuľky. Schéma jednotlivých tabuliek aj s reláciami na už existujúce tabuľky je na obrázku 4.4.

Tabuľka **subjectRecord** reprezentuje samotný prepísaný záznam. Okrem informácií, ktoré využíva systém DEMoS (autor prepisu, skóre, level užívateľa a iné) sa v tabuľke nachádzajú informácie o samotnom zázname (jazyk, register, umiestnenie na skene, panstvo, adresa a ďalšie).

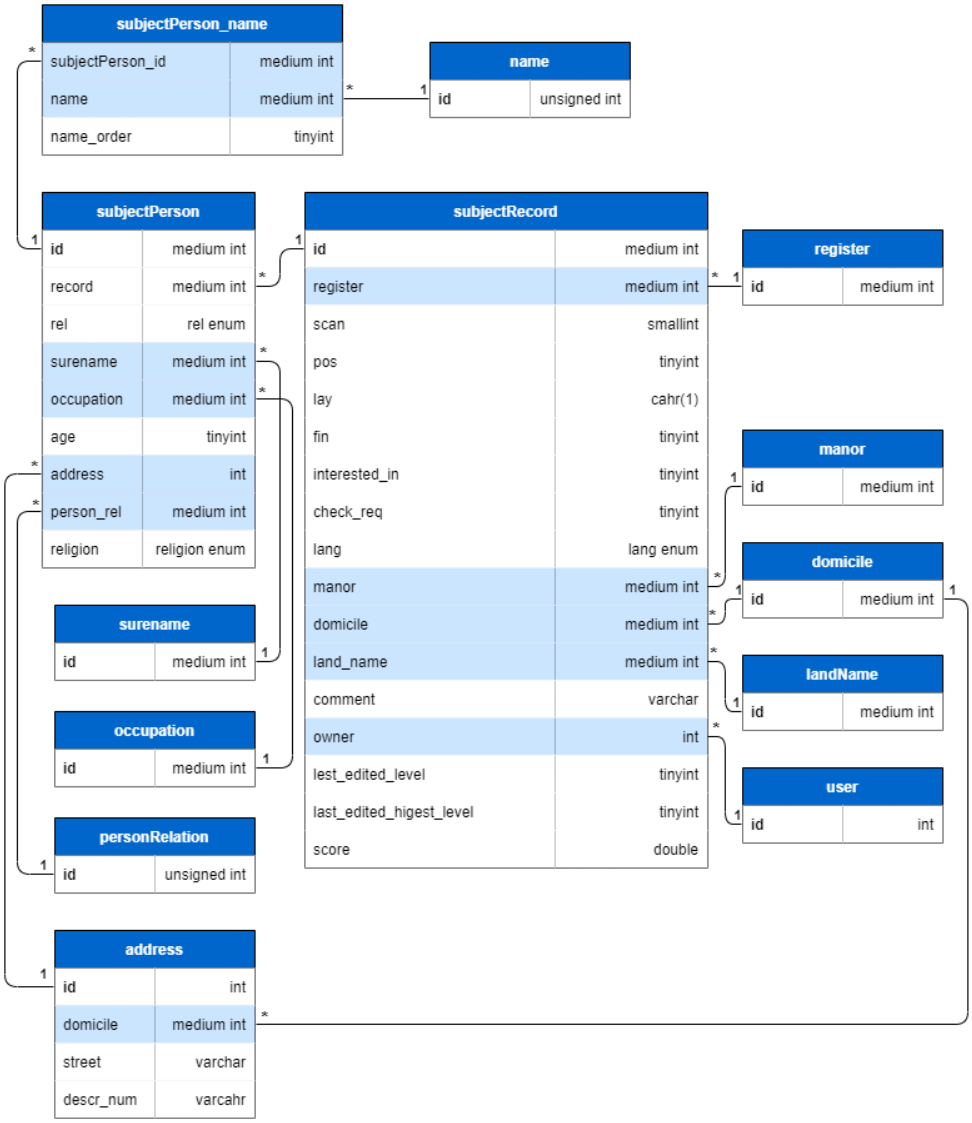
Druhá tabuľka, **subjectPerson**, obsahuje údaje o osobách, ktoré sú s daným záznamom spojené. Tabuľka obsahuje údaje, ako napríklad meno, povolanie, vek, náboženstvo a iné. V systéme je taktiež možné uchovať aj záznamy o príbuzných osobách daného poddaného. Jednotlivé osoby bude odlišovať stĺpec **rel**, ktorý označuje vzťah osoby k danému záznam. Pre soupis poddaných dle víry bude možné uložiť nasledujúce vzťahy:

- **subject** - označuje osobu, ku ktorej sa daný záznam viaže
- **relative_1, relative_2, relative_3, relative_4** - označuje príbuzenské osoby, systém môže evidovať až 4 takéto osoby

Na uloženie vzťahov príbuzenských osôb slúži tabuľka **personRelation** a odkazuje na ňu stĺpec **person_rel**. Táto tabuľka už bola v systéme vytvorená. Tretia tabuľka, **subjectPerson_name**, obsahuje jednotlivé mená danej osoby a ich poradie. Mimo spomínaných tabuliek sme využili tabuľky, ktoré už boli v systéme navrhnuté a uchovávajú časť údajov.

#### Popis niektorých stĺpcov tabuľky subjectRecord

- **interested_in** - označuje, či má autor o záznam záujem
- **check_req** - označuje, či ma autor prosí o kontrolu
- **manor** - odkaz do tabuľky **manor**, ktorá uchováva názvy panstiev
- **land_name** - odkaz do tabuľky **landName**, ktorá uchováva názvy gruntov




Obr. 4.4: Schéma tabuliek pre soupis poddaných dle víry

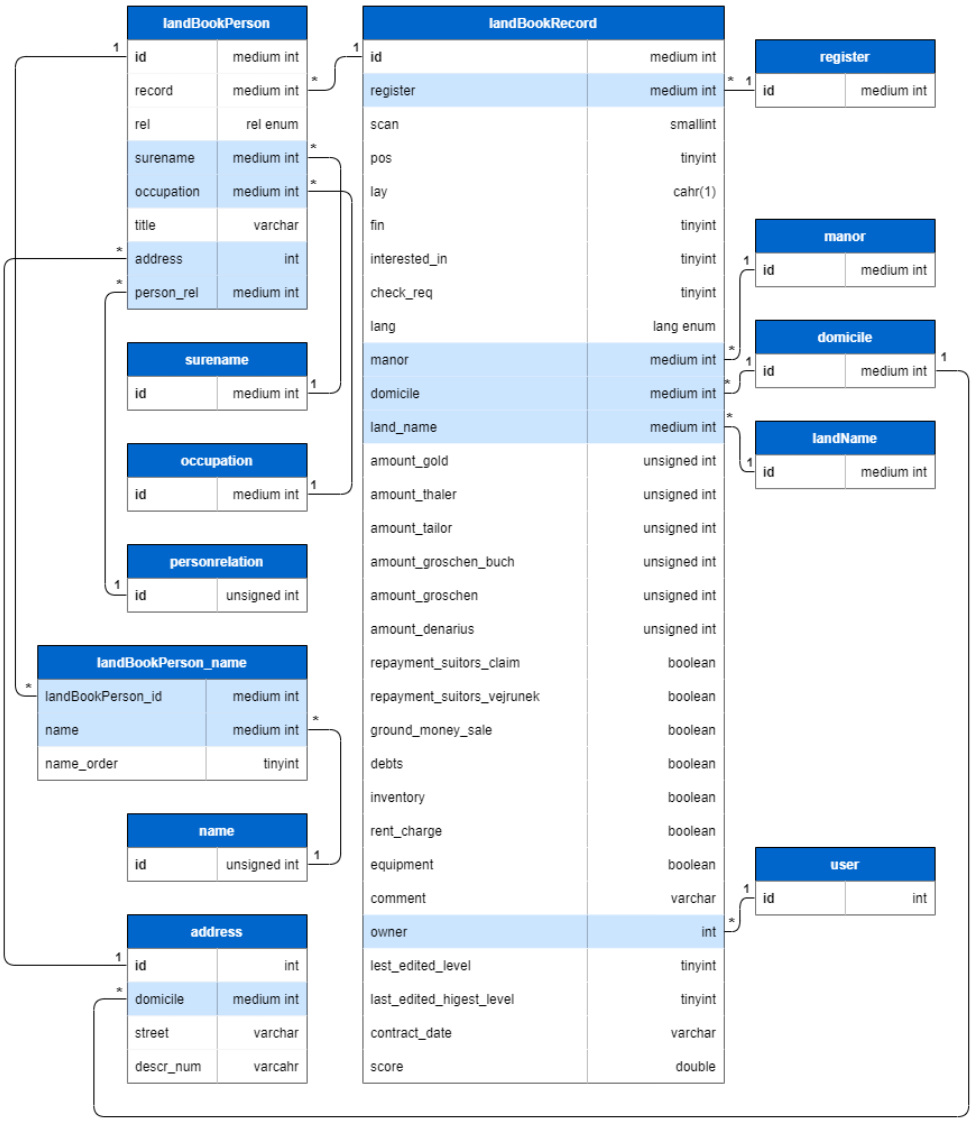
### Popis niektorých stĺpcov tabuľky subjectPerson

- **occupation** - odkaz do tabuľky **occupation**, ktorá uchováva jednotlivé povolania
- **religion** - vierovyznanie, môže nadobúdať hodnoty: catholic, protestant, jew, none

### 4.9.2 Pozemkové knihy

Pozemkové knihy sa vyvinuli z urbárov, o čom svedčí aj tá skutočnosť, že jeden z najstarších urbárov českých, urbár kláštora ostrovského z r. 1388, obsahuje jednotlivé knižničné zápisy vidieckeho majetku. Jedná sa o verejne prístupné zoznamy, do ktorých sa zapisovali nehnuteľnosti a práva vyplývajúce z ich vlastníctva, prípadne s tým spojené skutočnosti. Boli všeobecne dostupné a zapisovalo sa do nich pod dohľadom súdov.[11]

V databáze bolo potrebné pre prepis záznamov z pozemkových kníh vytvoriť 3 tabuľky. Schéma jednotlivých tabuliek aj s reláciami na už existujúce tabuľky je na obrázku 4.5.




Obr. 4.5: Schéma tabuliek pre pozemkové knihy

Tabuľka **landBookRecord** reprezentuje samotný prepísaný záznam. Tabuľka rovnako ako pri soupise poddaných dle víry obsahuje informácie potrebné pre systém DEMoS. Ďalej bude obsahovať informácie o pozemku alebo nehnuteľnosti, ako napríklad rozlohu, inventár dlhy a iné. Tabuľka **landBookPerson** a **landBookPerson_name** sú podobné ako pri tabuľkách so soupisem poddaných dle víry. Jediný rozdiel je ten, že neobsahujú údaje o náboženstve a je uchovávaný titul osoby. Jednotlivé typy osôb podľa vzťahu k záznamu sú nasledovné:

- **original_owner_1, original_owner_2** - označuje osobu alebo osoby, ktorým pozemok patrí
- **new_owner_1, new_owner_2** - označuje osobu alebo osoby, ktoré pozemok nadobudli

- **original_owner_1_rel**, **original_owner_2_rel** - označuje príbuzenské osoby k pôvodnému alebo aktuálnemu majiteľovi
- **new_owner_1_rel**, **new_owner_2_rel** - označuje príbuzenské osoby k novému majiteľovi

### Popis niektorých stĺpcov tabuľky landBookRecord

- **amount_gold** - označuje množstvo zlatých
- **amount_thaler** - označuje množstvo tolarov
- **amount_tailor** - označuje množstvo krejcarov
- **amount_groschen_buch** - označuje počet kop grošů
- **amount_groschen** - označuje množstvo grošů
- **amount_denarius** - označuje množstvo denárov
- **repayment_suitor_claim** - označuje, či išlo o zavdavek - drobný dar alebo symbolickú splátku
- **repayment_suitor_vejrunek** - označuje, či išlo o vejrunek - pravidelná splátka
- **ground_money_sale** - označuje, či išlo o predaj gruntovných peňazí
- **debts** - označuje, či sú k nehnuteľnosti viazané dlhy
- **inventory** - označuje, či sú k nehnuteľnosti viazaný inventár
- **rent_charge** - označuje, či ide o výměnek - prevod majetku
- **equipment** - označuje, či ide o výbavu

### 4.9.3 Sčítací operáty

Říšsky zákon o sčítaní z roku 1869 stanovil, že sčítanie obyvateľstva sa bude konať raz za desať rokov. Prvé sčítanie prebehlo k 31. decembru 1869 a všetky nasledujúce sčítania sa konali na polnoc z 31. decembra roku končiaceho nulou na 1. januára nasledujúceho roku. Sčítanie zahŕňalo prítomné aj domáce obyvateľstvo podľa príslušnosti k obci, vrátane dočasne alebo trvalo neprítomných osôb. Ministerstvo vnútra bolo zodpovedné za sčítanie, ktoré vykonávali obecné úrady. Sčítacou jednotkou bola domácnosť, a majiteľ domu alebo sčítací komisár vypĺňali sčítacie lístky. V roku 1880 bola pridaná otázka týkajúca sa obcovacieho jazyka a zisťovala sa aj gramotnosť obyvateľov. Od roku 1890 bolo povinné uvádzať presný dátum narodenia a zdokonalila sa evidencia povolania.[20]

V databáze bolo potrebné pre prepis záznamov zo sčítacích operatů vytvoriť 4 tabuľky. Schéma jednotlivých tabuliek aj s reláciami na už existujúce tabuľky je na obrázku 4.6.

Tabuľka **censusRecord** reprezentuje samotný prepísaný záznam. Tabuľka rovnako ako pri soupise poddaných dle víry obsahuje informácie potrebné pre systém DEMoS. Ďalej bude obsahovať informácie o nehnuteľnosti, počte osôb, informácie o biznise a ďalšie. Tabuľka **censusPerson_name** je podobné ako tabuľka pri soupisu poddaných dle víry. Tabuľka **censusPerson** obsahuje oproti pôvodným návrhom väčšie množstvo informácií ako

je napríklad gramotnosť, alebo adresa narodenia. Keďže tento historický prameň uchováva detailný prehľad o stave domácich zvierat, tieto údaje sú uložené v tabuľke **censusAnimal**. Jednotlivé typy osôb podľa vzťahu k záznamu sú nasledovné:

- **owner** - označuje majiteľa nehnuteľnosti
- **tenant** - označuje nájomníka

### Popis niektorých stĺpcov tabuľky censusRecord

- **year** - označuje rok sčítania
- **neighbour_buldings** - je krátky popis okolitých budov
- **secondary_purpose** - je krátky popis účelu užívania nehnuteľnosti
- **flat_inhabited** - označuje či je daná nehnuteľnosť obývaná
- **number_person** - je počet osôb žijúcich v nehnuteľnosti
- **business_occupation** - ak je nehnuteľnosť využívaná za účelom biznisu, tento stĺpec označuje povolanie, ktorým sa biznis zaoberá
- **business_profesional_status** - označuje bližší popis povolania
- **secondary_empl** - označuje skutočnosť, že majiteľ si privirába aj iným spôsobom ako svojim biznisom
- **absent_place** - označuje miesto, kde sa neprítomná osoba zdržuje
- **business_peasant_lands** - krátky popis pozemkov
- **businnes_realities** - krátky popis iných nehnuteľností

### Popis niektorých stĺpcov tabuľky censusPerson

- **rel** - označuje vzťah osoby k hlavnému záznamu
- **marital_status** - označuje manželský stav danej osoby
- **legitimate** - označuje lože danej osoby
- **home_law_address** - je odkazom do tabuľky **address** a označuje trvalý pobyt osoby
- **home_law_speach** - označuje obcovací jazik
- **read_write** - označuje gramotnosť osoby
- **blindness** - označuje slepú osobu
- **deafness** - označuje hluchú osobu
- **mental_illness** - označuje osobu z mentálnou poruchou
- **professional_status** - označuje postavenie v profesií

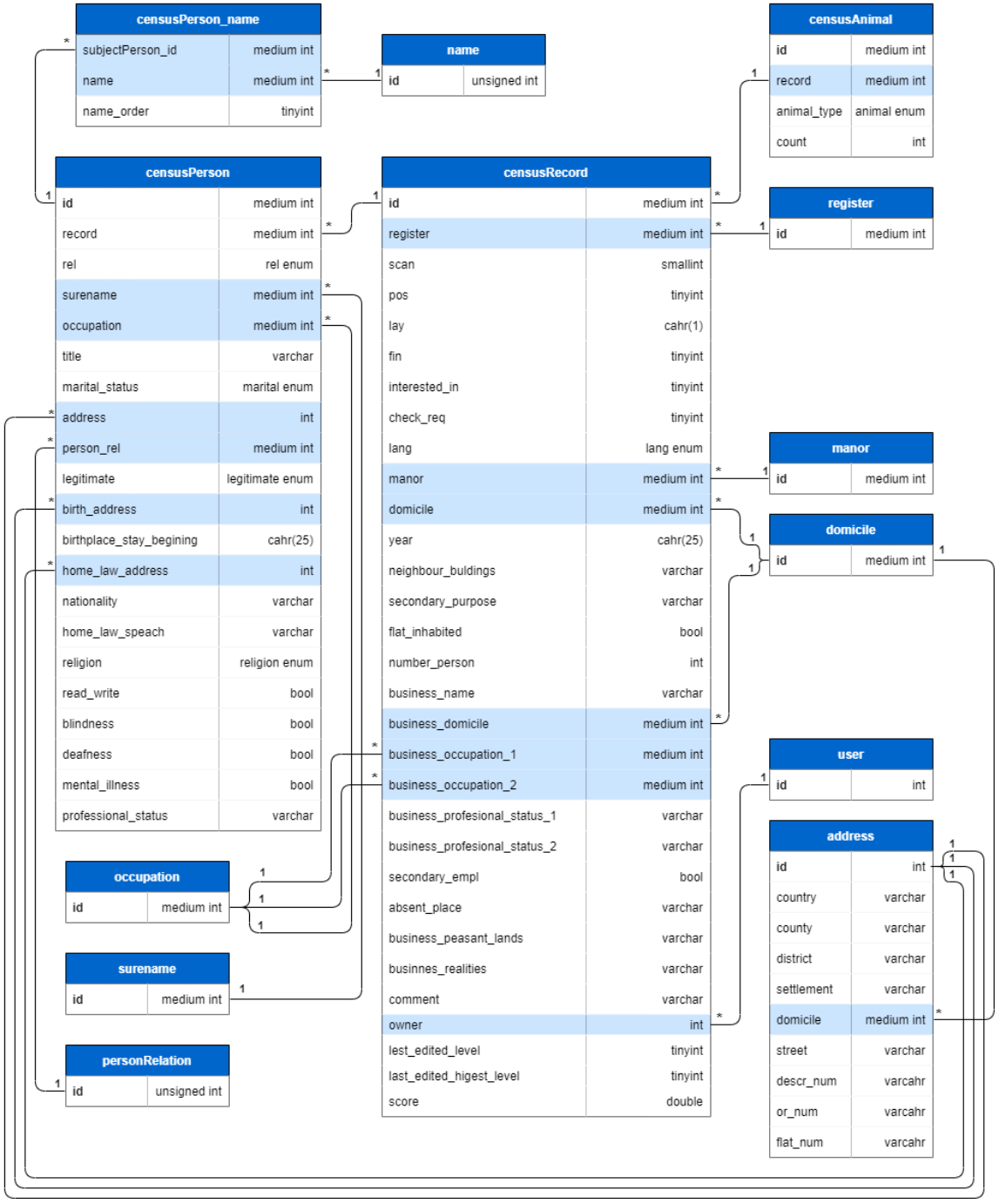
**Popis typov zvierat tabuľky censusAnimal**

- **hinny** - označuje mulice
- **mule** - označuje muly
- **donkey** - označuje somárov
- **goat** - označuje kozy bez rozdielu pohlavia a veku
- **heifer** - označuje jalovica do 1 roku života
- **ram** - označuje baranov starších ako 1 rok
- **sheep_female** - označuje ovce staršie ako 1 rok
- **wether** - označuje skopeca[4] staršieho ako 1 rok
- **piglet** - označuje svinky mladšie ako 3 mesiace
- **tread** - označuje prasatá mladšie ako 1 rok
- **boar** - označuje plemenné divé svine staršie ako 1 rok
- **swine** - označuje označuje plemenné svine staršie ako 1 rok
- **portk_other** - označuje iné prasatá staršie ako 1 rok
- **hive_movable** - označuje pohyblivé úly
- **hive_stacionary** - označuje ozačuje nepohyblivé úly
- **hive_mixed** - označuje označuje zmiešané úly
- **hen** - označuje sliepky
- **goose** - označuje husy
- **duck** - označuje kačky
- **poultry_others** - označuje inú hydinu

### 4.9.4 Pozície rámčekov pre jednotlivé skeny

Aby bolo možné evidovať rámčeky k skenom, bolo potrebné doplniť tabuľku **scanBorder**. Tabuľka je znázornená na obrázku 3.4. Ako primárny kľúč tejto tabuľky je zvolená kombinácia stĺpcov **reg_id**, **count**, **pos**, **lay**, ktoré tvoria unikátne označenie pre daný ramček. V tabuľke **registerScan** je podobný primárny kľúč, zložený z **reg_id**, **count**. Tieto tabuľky sú previazané pomocou cudzích kľúčov. Okrem iného tabuľka obsahuje súradnice ľavého horného okraja rámčeka a jeho šírku a výšku.

[4] skopec - vykastrovaný baran




Obr. 4.6: Schéma tabuliek pre sčítací operáty

# Kapitola 5

# Testovanie

Neoddeliteľnou súčasťou vývoja každej aplikácie je testovanie, ktoré ma za úlohu odhaliť chyby vzniknuté počas vývoja. Priebežné testy sme vykonávali už počas vývoja na lokálnom servery. Okrem iného sme použili ladiaci nástroj Tracy (3.1.3), ktorý je neoddeliteľnou súčasťou frameworku Nette. Priebežné výsledky práce boli analyzované spolu vedúcim práce a niekedy aj s náhodnými koncovými užívateľmi. Na staticku analýzu kódu sme použili PHPStan (PHP Static Analysis Tool), ktorý sa používa na detekciu chýb v kóde, ako sú typové chyby, volanie neexistujúcich metód alebo neznámych premenných. Pomocou tohto nástroja sme boli schopný identifikovať a opraviť potenciálne problémy ešte pred samotným spustením.

## 5.1 Užívateľské testovanie

Jedným z najbežnejších spôsobov hodnotenia užívateľského rozhrania je užívateľské testovanie, pri ktorom sa užívateľ usadí pri počítači a používa program. Počas tohto testovania užívateľ dostane sadu úloh, ktoré má vykonať v systéme, a jeho interakcia s vizuálnymi časťami aplikácie je sledovaná. Tento proces pomáha odhaliť problémy a nedostatky v užívateľskom rozhraní, ktoré mohli byť pri vývoji prehliadnuté alebo podcenené. Užívateľ má iný pohľad na systém než vývojár, ktorý ho implementoval a dobre ho pozná. Môže vyzdvihnúť problémy s funkčnosťou alebo neprehľadnosťou užívateľského rozhrania, ktoré by mohli uniknúť pozornosti vývojára.

Takéto testovanie by bolo vhodné ak by práca kládla dôraz na návrh a implementáciu užívateľského rozhrania. Nakoľko cieľom tejto práce bolo odladiť už fungujúci systém a teda nedošlo k vývoju nového užívateľského rozhrania. Drobné chyby užívateľského rozhrania boli detekované pri našom prvom kontakte so systémom, kedy sme ešte nepoznali rozsiahli kontext projektu. Ďalšie chyby boli identifikované pri doterajšom používaní aplikácie a preto nepovažujeme za potrebné zvoliť tento typ testovania.

## 5.2 Testovanie vkladania údajov

Po dokončení implementácie sme systém otestovali na vkladanie údajov a úpravu záznamov. Pri tomto testovaní sme sa zamerali na odhalenie chýb pri zápise do databáze. Pri vypĺňovaní polí sme sa zamerali okrem iného aj na frontendovú validáciu dátumov. Pri vytváraní nových záznamov sme otestovali následné scenáre:

1. záznam vyplnený iba povinnými údajmi

2. plne vyplnený záznam

Pri prvom scenári sme vypĺňali iba základné údaje ako je poradie skenu, poradie záznamu, pozícia záznamu a jazyk. Pri druhom scenári sme volili náhodné mená a názvy. Každý formulár sme vyplnili 2 razy a zamerali sme sa na to, či sa údaje uložia správne do databáze.

Formulár pre úpravu záznamu je zhodný s tým na vytvorenie nového. Na ich otestovanie sme použili následujúce scenáre:

1. postupná úprava všetkých formulárových polí

2. zmazanie všetkých formulárových polí

V týchto dvoch scenároch sme otestovali, že jednotlivé polia formuláru sa dajú editovať alebo zmazať, každú zmenu sme znovu pozorne sledovali v databáze.

Testovanie odhalilo niekoľko drobných chýb, prevažne prekleпov, ktoré sme následne odstránili.

# Kapitola 6

# Záver

Cieľom práce bola komplexná analýza systému ako celku so zameraním na funkčnosť a spoľahlivosť s následným odstránením odhalených chýb. Účelom tejto analýzy bolo zistiť či je systém funkčný, teda či fungujú jednotlivé časti. Analýzou sme zistili vážne nedostatky v konzistencii systému a v jeho funkcionalite. Výsledkom je tak rozsiahli refaktoring zdrojového kódu, ktorým sa nám podarilo eliminovať množstvo chýb. Okrem toho sa nám podarilo systém rozšíriť o chýbajúcu funkcionalitu, predovšetkým sa jednalo o doplnenie chýbajúcich historických prameňov. Do systému sme doplnili soupis poddaných dle víry, pozemkové knihy a sčítaci operáty. Taktiež sme upravili užívateľské rozhranie, tak aby bolo užívateľský prívetivejšie. Návrh na úpravu a rozšírenie pôvodného systému vznikol na základe naštudovaných poznatkov z oblasti genealógie. Systém DEMoS tak podporuje prepis všetkých historických prameňov, ktoré sa vyskytujú na území Českej republiky.

Projekt DEMoS je pomerne veľký projekt a je na nom ešte veľa práce. V systéme je veľa funkcionality vyvinutej na prepis matričných záznamov, táto funkcionalita by sa časom mohla rozšíriť aj na ostatné historické pramene. Priestor pre zlepšenie tiež vidím vo výkone aplikácie, ktorý by sa dal zvýšiť za použitia bundlingu. Tiež by bolo potrebné sa zamerať na jazykovú stránku celého systému, preklady nie sú správne nastavené a taktiež by sa systém mohol rozšíriť o ďalšie jazykové mutácie.

# Literatúra


[1] *Digitalizace v archivech* [online]. Česká genealogická a heraldická společnost v Praze [cit. 6.5.2023]. Dostupné z: http://www.genealogie.cz/aktivity/digitalizace/.

[2] *Dokumentácia Doctrine* [online]. [cit. 4.5.2023]. Dostupné z: https://symfony.com/doc/current/doctrine.html.

[3] *Dokumentácia Latte* [online]. [cit. 4.5.2023]. Dostupné z: https://latte.nette.org/cs/.

[4] *Dokumentácia Nette: 7 důvodů, proč používat Nette* [online]. [cit. 2.5.2023]. Dostupné z: https://nette.org/cs/10-reasons-why-nette.

[5] *Dokumentácia Nette: Aplikácia Nette* [online]. [cit. 3.5.2023]. Dostupné z: https://doc.nette.org/cs/application.

[6] *Dokumentácia Nette: Dependency Injection* [online]. [cit. 2.5.2023]. Dostupné z: https://doc.nette.org/cs/dependency-injection.

[7] *Dokumentácia Nette: Generované továrne* [online]. [cit. 3.5.2023]. Dostupné z: https://doc.nette.org/cs/dependency-injection/factory.

[8] *Dokumentácia Nette: Presentery* [online]. [cit. 3.5.2023]. Dostupné z: https://doc.nette.org/cs/application/presenters.

[9] *Download Composer* [online]. GetComposer [cit. 2.5.2023]. Dostupné z: https://getcomposer.org/download/.

[10] *Soupis poddaných podle víry* [online]. statistika&my [cit. 6.5.2023]. Dostupné z: https://www.statistikaamy.cz/pribeh-statistiky/1651-soupis-poddanych-podle-viry/.

[11] *VEŘEJNÁ KNIHA* [online]. webpark [cit. 7.5.2023]. Dostupné z: https://krovak.webpark.cz/katastr/vk.htm.

[12] *What Is jQuery? A Look At the Web's Most-Used JavaScript Library* [online]. kinsta, 20. marca 2023 [cit. 7.5.2023]. Dostupné z: https://kinsta.com/knowledgebase/what-is-jquery/.

[13] Brent. *PHP version stats: January, 2023* [online]. upGrad, september 2022 [cit. 1.5.2023]. Dostupné z: https://stitcher.io/blog/php-version-stats-january-2023.

[14] Chris, K. *What is PHP? The PHP Programming Language Meaning Explained* [online]. freeCodeCamp, 30. augusta 2021 [cit. 1.5.2023]. Dostupné z: https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/.

[15] Czernín, D. *Uživatelské rozhraní pro přepis matrik*. Brno, CZ, 2018. [cit. 8.5.2023]. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace?zp_id=114694.

[16] Gillis, A. S. *Dependency injection* [online]. TechTarget, marec 2023 [cit. 7.5.2023]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/dependency-injection.

[17] Grudl, D. *Nette Framework: MVC & MVP* [online]. zdroják.cz, 24. marca 2009 [cit. 5.5.2023]. Dostupné z: https://zdrojak.cz/clanky/nette-framework-mvc-mvp/.

[18] Heidi, E. *OOPS Concepts in PHP | Object Oriented Programming in PHP* [online]. DigitalOcean, 25. marca 2021 [cit. 1.5.2023]. Dostupné z: https://www.digitalocean.com/community/tutorials/what-is-composer.

[19] Malina, J. *Encyklopedie antropologie*. 2011 [cit. 8.5.2023]. Dostupné z: https://is.muni.cz/elportal/?id=962501.

[20] Monika. *Sčítání lidu* [online]. neznamo, 11. októbra 2019 [cit. 7.5.2023]. Dostupné z: @webpage{census,author="Monika",title="SÄDÃŋtÃąnÃŋlidu",howpublished="online",publisher="neznamo",url="https://www.neznamo.cz/genealogie/scitani-lidu/",year="2019",month=10,day=11,cited="6.5.2023"}.

[21] Olawanle, J. *What is a Framework? Software Frameworks Definition* [online]. freeCodeCamp, 25. marca 2021 [cit. 2.5.2023]. Dostupné z: https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/.

[22] Pecháček, J. *Genealogické diagramy aneb co je to vlastně rodokmen?* [online]. odkudjsme [cit. 6.5.2023]. Dostupné z: https://www.neznamo.cz/genealogie/scitani-lidu/.

[23] Skřebská, Z. *GENEALOGIE A TVORBA RODOKMENŮ*. Brno, CZ, 2017. [cit. 7.5.2023]. Bakalárska práca. Masarykova univerzita, Přírodovědecká fakulta, Ústav antropologie. Dostupné z: https://is.muni.cz/th/mrrs2/Bakalarska_prace_-_Skrebska.pdf.

[24] Smejkal, P. *Co je AJAX?* [online]. Petr Smejkal [cit. 6.5.2023]. Dostupné z: https://www.petrsmejkal.cz/slovnik/ajax/.

[25] Vats, R. *OOPS Concepts in PHP | Object Oriented Programming in PHP* [online]. stitcher, január 2023 [cit. 1.5.2023]. Dostupné z: https://www.upgrad.com/blog/oops-concepts-in-php/.

[26] Vinther, D. *Position: stuck; — and a way to fix it* [online]. uxdesign, 5. februára 2019 [cit. 5.5.2023]. Dostupné z: https://uxdesign.cc/position-stuck-96c9f55d9526.

[27] Zatloukal, T. *Interaktivní GUI pro zadávání matričních záznamů*. Brno, CZ, 2019. [cit. 8.5.2023]. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis-file/22006/22006.pdf.

[28] Zola, A. *Bootstrap* [online]. TechTarger, august 2022 [cit. 5.5.2023]. Dostupné z: https://www.techtarget.com/whatis/definition/bootstrap.