Elektronický hlasovací systém

Lukáš Richter

Bakalářská práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** Nascanované zadání, strana 1 ***

*** Nascanované zadání, strana 2 ***

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářské práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne .....

podpis studenta

ABSTRAKT

Text abstraktu česky

Klíčová slova: Přehled klíčových slov

ABSTRACT

Text of the abstract

Keywords: Some keywords

Zde je místo pro případné poděkování, motto, úryvky knih, básní atp.

OBSAH

ÚVOD

První řádek prvního odstavce v kapitole či podkapitole se neodsazuje, ostatní ano. Vertikální odsazení mezy odstavci je typycké pro anglickou sazbu; czech babel toto respektuje, netřeba do textu přidávat jakékoliv explicitní formátování, viz ukázka sazby tohoto textu s následujícím odstavcem).

Formátování druhého odstavce. Text text text text text text text text text text text text.

I. TEORETICKÁ ČÁST

1 ELEKTRONICKÉ VOLBY

Na této stránce je k vidění způsob tvorby různých úrovní nadpisů.

1.1 Koncepce, výhody, nevýhody

Text

1.2 Požadavky, podmínky

Text

1.3 RSA Blind Signature

Text

2 POŽADAVKY NA FUNKČNOST

Níže následují ukázky vložení obrázku, tabulky a různorodých citací.

2.1 Obecné požadavky

2.2 Specifika provozu na UTB

2.3 Navrhované řešení

3 NÁVRH APLIKACE

Jádrem celé aplikace byl zvolen PHP framework Nette od českého vývojáře Davida Grudla. V konkurenci světových frameworků jako je Laravel nebo Symfony je velice oblíbený především v českém prostředí. Zcela jistě i díky kvalitní dokumentaci v češtině, pravidelným aktualizacím i aktivnímu diskuznímu fóru. Nette za velkými hráči rozhodně nezaostává a naopak přináší velice intuitivní způsob tvorby kvalitních, rychlých a bezpečných webových aplikací [**?**].

3.1 Architektura MVC

Nette patří do skupiny architektonických vzorů známých jako MVC (Model View Controller), přesněji MVP (Model View Presenter). Jako první popsal MVC v roce 1979
Trygve Reenskaug pro programovací jazyk Smalltalk [**?**]. Základním principem je rozdělení systému do tří samostatných částí - data jako Model a vstup a výstup jako Controller resp. View. S vývojem počítačů ustupovala potřeba tohoto dělení, jelikož jedna komponenta systému již uměla obsloužit vstup i výstup zároveň. S příchodem a rozmachem internetu se MVC vrátilo a zatím zůstává [**?**].

V kontextu webové aplikace chápeme Model jako data a jejich obsluhu, View jako zobrazení těchto dat uživateli a Controller zpracovává uživatelské vstupy, manipuluje s Modelem a aktivuje View. Uživatelské rozhraní je v tomto podání tedy kombinací View a Controlleru. Současné frameworky nejčastěji kombinují vzory Front Controller (obsluha HTTP požadavku) a Page Controller (samotná logika konrétní části aplikace) [**?**].

Variantu MVP (Model View Presenter) v současném podání popisuje Fowler[**?**] jako vzor Passive View. Dochází k těsnější vazbě Controlleru (resp. Presenteru) a View a zároveň je Model izolován od View. Například v Nette neexistuje obdoba Front Controlleru, už z URL adresy totiž aplikace pozná, který Presenter i jeho metoda je volána. Logika Front Controlleru se tedy rozpustila mezi View a Page Controller, kterému se říká Presenter.

3.2 Nette framework

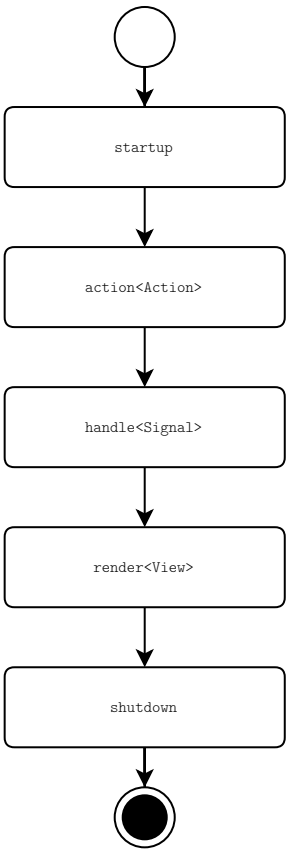
Jak již bylo řečeno, jedná se architektonický vzor MVP. Jednotlivé části je potřeba chápat jako abstraktní vrstvy, nelze si pod nimi představit konkrétní (PHP) třídy. Jedním důvodem je možné prolínání vrstev v rámci jedné třídy, tím druhým a závažnějším je pak nepochopení celého principu MVC/P architektury. Tím je myšleno domění mnohých (začínajících) programátorů, že Model je jeden konkrétní objekt

(entita) [zdroj - je nutny?], přičemž modelová vrstva jsou nejen entity ale i business (nebo doménová) logika a dohromady tvoří tuto modelovou vrstvu. Pokud bude v této práci zmíněn **Presenter**, je tím myšlena konkrétní třída nebo skupina tříd nikoliv vrstva.

3.2.1 Třída Presenter

Presenter přijímá objekt `Nette\Application\Request`, který představuje HTTP požadavek a pomocí něho určí, jaké konkrétní metody je potřeba zavolat.

Obrázek 3.1 přehledně popisuje sled volání jednotlivých metod, přičemž jsou všechny nepovinné. Vynecháním definic všech metod by došlo pouze k odeslání statického obsahu šablony.



Obrázek 3.1 Životní cyklus presenteru

Požadavek v Nette je tvořen kombinací `Presenter:Action`. *Signál* rozšiřuje základní požadavek a volá se vždy současně s aktuálním presenterem a akcí. Nejčastěji se signály využívají pro AJAXové požadavky nebo v komponentách, což jsou samostatné znovupoužitelné objekty. Samotný presenter je potomkem komponenty, z toho vyplývá, že komponenty mohou s View komunikovat napřímo pouze díky signálům. V obrázku nejsou uvedeny metody `beforeRender()` a `afterRender()`, které společně se `startup()` a `shutdown()` nemají vazbu na *akci*

nebo *signál* a mohou být v rámci Presenteru definovány právě jednou, slouží k definici společného chování napříč různými akcemi[**?**].

3.2.2 Routování

Veškeré HTTP požadavky klienta míří na soubor `www/index.php` , zde se inicialzuje prostředí Nette, požadavek se přeloží do objektu `Nette\Application\Request` a vyvolá se příslušný Presenter. Proces překládání HTTP požadavků, resp. překlad URL se běžně u PHP frameworků označuje jako routování. Router umí URL adresu nejen rozložit ale také složit (neboli vytvářet odkazy). Maska routy říká routeru, jak přeložit URL adresu na tvar `Presenter:Action`, případně složitější tvary[**?**]. Základní instalace Nette obsahuje jednu jedinou routu, která je vidět na výňatku kódu 3.1, tato routa převede URL tvaru `https://domain.com/presenter/action` na požadavek tvaru `Presenter:Action`, přičemž parametr `id` se předává jako argument metodá action a render příslušného presenteru. Pokud není v URL přítomná část presenter nebo action, doplní se o výchozí nastavení, zde `Homepage:default`. Tabulka 3.1 obsahuje příklady překladů pomocí této základní routy.

```php
<?php

declare(strict_types=1);

namespace App\Router;

use Nette;
use Nette\Application\Routers\RouteList;


final class RouterFactory
{
  use Nette\StaticClass;

  public static function createRouter(): RouteList
  {
    $router = new RouteList;
    $router->addRoute('<presenter>/<action>[/<id>]', 'Homepage:default');
    return $router;
  }
}
```

Fragment zdrojového kódu 3.1 Základní routa v Nette

Latte Šablonovací systém od vývojáře Nette, představuje výborné spojení s Nette. ...

Tabulka 3.1 Příklady routování

URL adresa	Nette požadavek
https://domain.com/	Homepage:default
https://domain.com/homepage/about	Homepage:about
https://domain.com/article/view/12	Article:view, id = 12

3.3 Domain Driven Design

Při návrhu aplikace bylo využito zásad Domain Driven Desingu (dále též DDD), který uvedl Eric Evans ve své knize *Domain-driven Design: Tackling Complexity in the Heart of Software* [**?**]. Tento styl vývoje software si klade za cíl řešit návrh komplexního řešení pomocí modelu obchodní domény. Úzká polupráce klienta (doménoví experti) a vývojářů (techničtí experti) probíhá za pomoci společného a jednotného jazyka (*Ubiquitous Language*). Aplikace by měla být rozdělena do několika základních vrstev a to konkrétně [**?**]:

- **Prezentační vrstva** - přenáší informace uživateli a obsluhuje jeho požadavky
- **Aplikační vrstva** - koordinuje práci ostatních objektů, neobsahuje business logiku
- **Doménová vrstva** - hlavní část DDD, která obsahuje doménové objekty a kompletní business logiku
- **Vrstva infrakstruktury** - poskytuje prostředky ostatním vrstvám (komunikace, perzistence aj.)

Při srovnání koncepcí vrstev podle MVC/P a DDD lze říci, že se vhodně překrývají, pokud je zajištěno, že Controller neobsauje logiku doménové vrstvy. V případě MVP pak částečně dochází k prolínání aplikační a prezentační vrstvy, což podle Evanse nehraje zásadní roli, tou je separace doménové vrstvy [**?**]. Dále Evans definuje základními stavební bloky doménové vrstvy [**?**]:

- **Value Object** - neměnný (immutable) objekt, který reprezentuje nějakou hodnotu / vlastnost. Může to být telefonní číslo ale i poštovní adresa skládající se z několika částí (členských proměnných).
- **Entity** - základní objekt, který je jednoznačně identifikován svojí *identitou*, jeho vlastnosti mohou být reprezentovány value objekty nebo dalšími entitami, z čehož vznikají agregáty.
- **Aggregate** - skupina entit, která v doméně představuje celek. *Root Aggregate* pak představuje vstupní bod pro okolní objekty k agregátovi. Objekty uvnitř agregátu

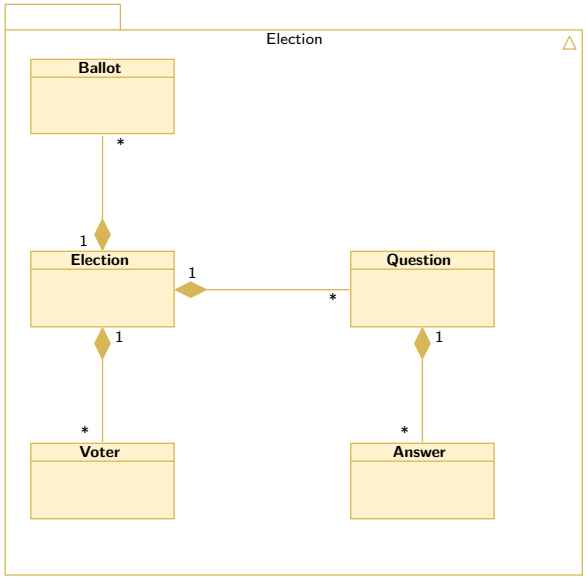
mohou mít libovolné vazby mezi sebou, ale ne na okolní objekty. Převážná část business logiky by se měla odehrávat právě zde.

- **Module** - logické propojení (v kontextu domény) entit a agregátů vytváří moduly / balíčky.
- **Factory** - továrny na objekty zapouzdřují proces vytváření nových objektů. Může to být metoda agregátu, která vytváří instance jednotlivých entit a value objektů nebo samostatná třída.
- **Service** - pokud nějaká operace nedává smysl v rámci jednoho objektu, může být zapouzdřena do samostatného objektu služby.
- **Repository** - získávání objektů, jejich perzistence (ukládání, mazání) je zapouzdřeno do samostatných repozitářů.

3.4 Entity

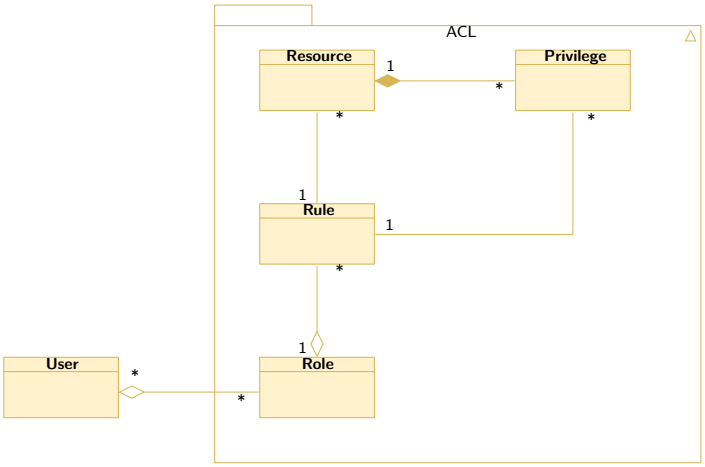
Na základě shromážděných požadavků na aplikaci v kapitole [odkaz na kapitolu]byl sestaven obecný model kritických částí aplikace. Základem volební aplikace je samozřejmě model hlasování. Entita `Election` představuje kořen stejnojmeného agregátu (*Aggregate Root*). Vazby mezi jednotlivými entitami jsou znázorněny jako diagram modelu v obrázku 3.2. Jednotlivými entitami tohoto agregátu jsou:

- **Election** kořen agregátu představující jedny konkrétní volby / hlasování
- **Question** ve volbách představuje volenou pozici, v obecném hlasování jednu otázku
- **Answer** je množina kandidátů, resp. odpovědí na otázku
- **Voter** zahrnuje všechny oprávněné voliče
- **Ballot** jsou všechny odevzdané hlasovací lístky v daných volbách / hlasování



Obrázek 3.2 Model objektů balíčku Election

Druhou zásadní částí aplikace je systém pro správu přístupu uživatelů (ACL). Nejjednodušší implementací takového systému je přiřazení oprávnění pomocí statické konfigurace. Tento přístup podporuje Nette bez nutnosti jakéhokoli dalšího rozšiřování o vlastní správu oprávnění. Nicméně takový přístup značně limituje flexibilitu aplikace, jelikož se jakákoli změna musí ručně zapsat do konfigurace, která bývá většinou uložena na serveru ve formě souboru. Z tohoto důvodu byl namodelován vlastní ACL systém.



Obrázek 3.3 Model objektů balíčku ACL

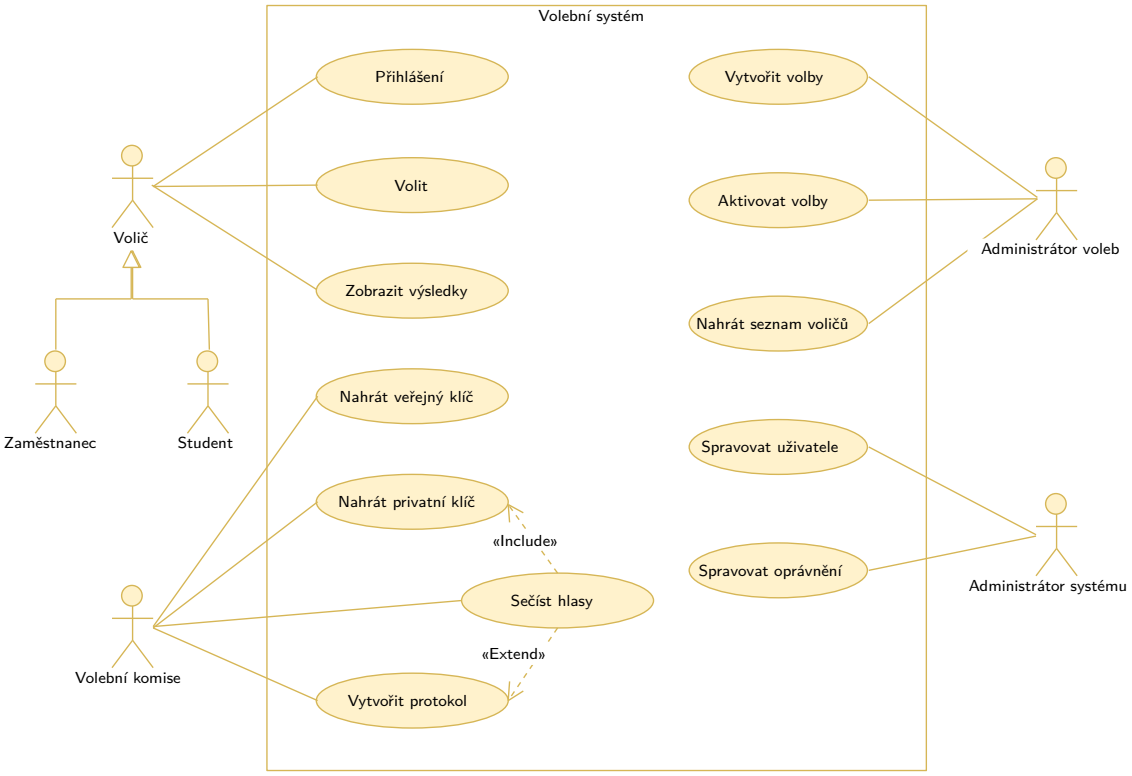
V tomto případě je kořenem agregátu entita `Role` symbolizující jednu roli uživatele. Role může mít nastavena pravidla `Rule`, která budou řídit přístup k prostředkům `Resource` a akcím `Privilege` na nich vykonávaných. Každé pravidlo je kombinací právě jednoho prostředku a jedné akce. Pravidlo zároveň určuje, jestli je pro danou roli tato akce povolena nebo zakázána (*allow / deny*). Tabluka 3.2 ukazuje příklady možného nastavení tohoto systému.

Tabulka 3.2 Příklady nastavení ACL

Role	Prostředek	Akce	Pravidlo
Student	Election	View	allow
Student	Election	Vote	allow
Student	Election	Delete	**deny**
Komise	Election	Count	allow
Administrator	Election	Activate	allow
SuperAdmin	User	Create	allow

3.5 Průchod voliče webem

Jako další byl vytvořen zjednodušený případ užití celého systému, podle stanovanených požadavků, především s ohledem na zvolený systém anonymizace hlasovacích lístků, tak jak byl popsán v kapitole [odkaz na kapitolu].



Obrázek 3.4 Případ užití systému

II. PRAKTICKÁ ČÁST

4 IMPLEMENTACE NÁVRHU

Konkrétní implementace modelu popsaného v teoretické části probíhala postupně a vyvíjela se. Některé způsoby se po čase ukázaly jako nevhodné či nedokonalé a bylo potřeba je upravit resp. přepracovat tak, aby aplikace jako celek byla funkční. Během vývoje aplikace bylo změněno i IDE z Visual Studio Code na PHPstorm, což je místy vidět v rozdílných PHPdoc komentářích zdrojového kódu.

4.1 Členění aplikace

Jednou ze změn, které se objevily jako vhodné v průběhu vývoje bylo rozdělení aplikace na dvě části, v kontextu Content Management Systémů běžně označované jako Frontend a Backend. Tyto dvě části by na sobě měly být nezávislé, tedy jedna nepotřebuje vědět o (ne)existenci té druhé a umí svoje úkoly provést zcela samostatně. Frontend představuje tu část, která je veřejně dostupná, Backend označuje neveřejné administrační rozhraní aplikace. Tohoto rozdělení se zároveň využilo ke zvýšení bezpečnosti aplikace, jak je vysvětleno v části 4.1.1.

Zjednodušená adresářová struktura je zobrazena v příloze 1. V adresáři `app/` se nachází zdrojové kódy rozdělené podle jejich účelu. Presentery jsou společně se šablonami v příslušných adresářích (Backend, Frontend a Core). Core obsahuje presentery společné pro Frontend i Backend, což jsou momentálně pouze presentery pro zpracování chybových hlášení. Jednotlivé adresáře jsou popsány níže.

- **Backend** - obsahuje Presentery, šablony a pomocné třídy využité v Backendové části
- **Config** - konfigurační soubory aplikace
- **Core** - obsahuje Presentery, šablony a pomocné třídy využité napříč celou aplikací
- **Forms** - definice a továrny pro složitější formuláře
- **Frontend** - obsahuje Presentery a šablony využité ve Frontendové části
- **Models** - obsahuje modelovou vrstvu
- **Repositories** - všechny repozitáře pro komunikaci s modelovou vrstvou
- **Router** - definice routování

4.1.1 Fyzické oddělení

Prvním krokem zabezpečení neveřejné části webu je samozřejmě omezení přístupu heslem přímo v aplikaci. Přihlašovací formulář je nicméně stále veřejný a kdokoli s odkazem na přihlašovací stránku se může pokoušet o přihlášení. Druhým krokem tedy může být omezení přístupu pomocí IP adres (například pouze na adresy vnitřní sítě UTB) a to pomocí nastavení HTTP serveru souborem `.htaccess` nebo v konfiguraci `virtualhost` . Toto nastavení je přenecháno ke zvážení tomu, kdo bude zodpovědný za instalaci a nastavení serveru.

Aby veškeré odkazy v aplikaci fungovaly a aby Nette vědělo kam má směřovat požadavky, bylo potřeba upravit základní routování popsané v části 3.2.2. Routy podporují tzv. moduly, které slouží přesně k takovému rozdělení aplikace na několik oddělených částí. Pro každý modul je možné definovat vlastní routy, seskupení rout do modulů se provádí voláním metody `withModule(string $module)` třídy `RouteList`.

```
1  public static function createRouter(): RouteList
2  {
3    $router = new RouteList;
4    if ($_SERVER['SERVER_NAME'] == 'admin.volby.l') {
5      $router->withModule('Backend')
6        ->addRoute('//admin.%domain%/prihlasit', 'Sign:in')
7        ->addRoute('//admin.%domain%/odhlasit', 'Sign:out')
8        ->addRoute('//admin.%domain%/<presenter>/<action>[/<id>]',
         ↪  'Homepage:default');
9    }
10   if ($_SERVER['SERVER_NAME'] == 'admin.volby.lukasrichter.eu') {
11     $router->withModule('Backend')
12       ->addRoute('//admin.volby.%domain%/prihlasit', 'Sign:in')
13       ->addRoute('//admin.volby.%domain%/odhlasit', 'Sign:out')
14       ->addRoute('//admin.volby.%domain%/<presenter>/<action>[/<id>]',
         ↪  'Homepage:default');
15   }
16   $router->withModule('Frontend')
17     ->addRoute('/prihlasit', 'Sign:in')
18     ->addRoute('/odhlasit', 'Sign:out')
19     ->addRoute('/<presenter>/<action>[/<id>]', 'Homepage:default');
20   return $router;
21 }
```

Fragment zdrojového kódu 4.1 Upravená routa v Nette

Rozšířené nastavení routování je patrné z fragmentu 4.1. Obsahuje nastavení pro lokální testování (admin.volby.l) i simulaci produkčního prostředí na VPS serveru v internetu (admin.volby.lukasrichter.eu). Pro Backendový modul bylo potřeba rozlišit jednotlivá prostředí podle názvu serveru kvůli použití domény čtvrtého řádu, se

kterou si Router neporadil. Pro Frontendový modul toto nebylo potřeba, doména třetího řádu (volby.lukasrichter.eu) fungovala v pořádku. Zápis `addRoute('/prihlasit', 'Sign:in')` definuje alias pro akci `in` presenteru `SignPresenter` , která je standardně dostupná pod adresou `/sign/in` .

S tímto nastavením jsou jednotlivé části aplikace dostupné ze samostaných domén (např. admin.volby.utb.cz a volby.utb.cz), které nemusí být fyzicky na stejném serveru. Právě toto dokáže podstatně zvýšit bezpečnost aplikace. Frontendová část (volby.utb.cz) je umístěna na bežném veřejně přístupném serveru, zatímco Backendová část je umístěna na serveru, který nemusí být vůbec dostupný z internetu. Samozřejmě může být rozdílná i samotná doména nižšího řádu, za předpokladu, že jsou správně nastaveny DNS záznamy.

A jelikož jsou obě části na sobě nezávislé, není nutné na veřejně dostupném Frontendu umisťovat Backendový kód, který obsahuje například zpracování, dešifrování a počítání odevzdaných hlasů, ale i aktivaci a mazání celých voleb. V případě útoku na aplikaci s cílem kompromitovat nebo ovlivnit volby, nemají útočníci možnost tento kód spustit. Museli by tedy útočit přímo na databázový server, jehož zabezpečení je v komptenci administrátora serveru.

5 DOMÉNOVÁ VRSTVA

Celý proces získání konkrétní entity pro potřeby aplikační vrstvy je postaven na několika návrhových vzorech. Správné užití návrhových vzorů umožní zapouzdřit chování jednotlivých tříd, nebudou vytvářena těsná propojení jednotlivých tříd a celý zdrojový kód bude flexibilnější. Cílem tohoto přístupu je zjednodušení případných budoucích úprav aplikace. Těsné provázání tříd aplikační a databázové vrstvy sice znamená méně náročnou práci při první implementaci, ale o to je náročnější kód v budoucnosti upravit.

Příkladem může být objekt – entita, který se umí perzistovat, tj. je přímo závislý na konkrétním úložišti. Při změně úložiště je pak potřeba upravit všechny takové objekty.

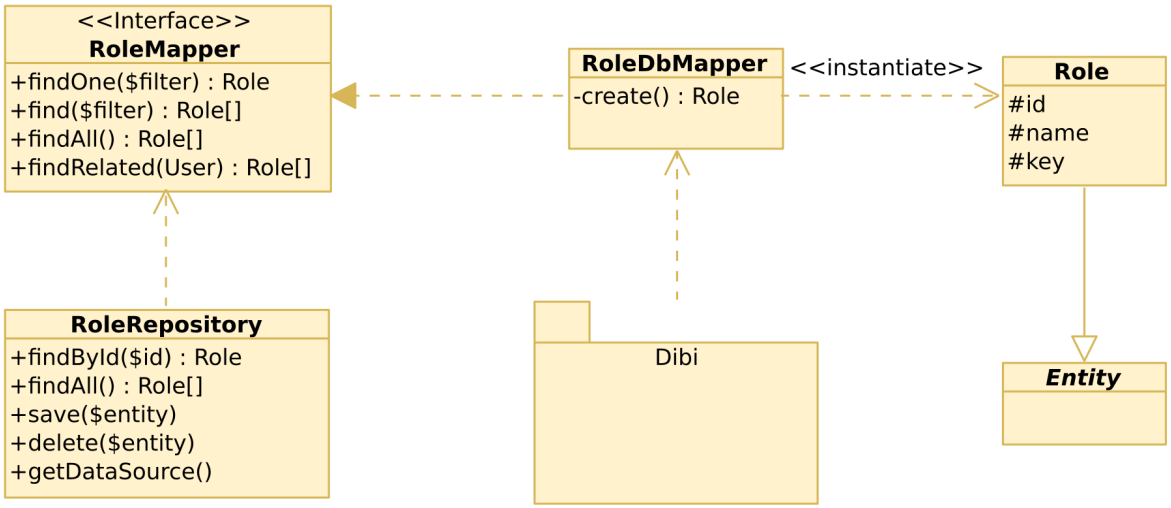
5.1 Modelová vrstva

Z pohledu architektury MVC reprezentuje modelová vrstva data a manipulaci s nimi a v aplikaci se výrazně překrývá s doménovou vrstvou z pohledu DDD. Způsob implementace byl zvolen tak, aby zbytek aplikace nebyl pevně spojen se způsobem získávání a ukládání dat (entit).

Single Responsibility Principle (Princip jedné odpovědnosti) zavedený Robertem C. Martinem udává, že "třída by měla mít pouze jeden důvod ke změně"[1)] . Změnou je zde myšleno přepracování kódu. Pokud má třída pouze jednu odpovědnost, pouze ta může vyvolat nutnost změny kódu. Entita má odpovědnost podávat o sobě informace, její odpovědností není, jak je vytvořena, jak a jestli vůbec je perzistována v databázi či jinde atd. Vytváření entit by měla mít na starosti třída typu `Factory`, převedení dat z úložiště do formátu, kterému továrna rozumí je úkolem pro `Data Mapper`. Získání entit pro potřeby aplikace je práce pro `Repository`.

Získávání a manipulace s entitami byla implementována pomocí návrhových vzorů Data Mapper a Repository. Aplikační vrstva (především Presentery) získává jako závislost repozitáře (třídy typu Repository), které jí poskytují požadované entity nebo kolekce entit. V souladu se SRP Presenter nezajímá, jakým způsobem jsou entity získávány, k jeho odpovědnosti to nepatří. Repozitáře vědí, že rozhraní Data mapper umí poskytovat entity bez ohledu na to, kde a jak je konkrétní entita uložena (v paměti, souboru, databázi či jinde). V aplikaci je pouze jedna implementace a to `DbDataMapper`. Data mapper pomocí databázového adaptéru Dibi odesílá požadavky na databázový server. Vytváření entit je řešeno částečně továrními metodami data mapperů a částečně samotnými továrními třídami v závislosti na složitosti operace. Ideální by ovšem bylo striktní oddělení do samostatných tříd.

[1)] A class should have only one reason to change[**?**]



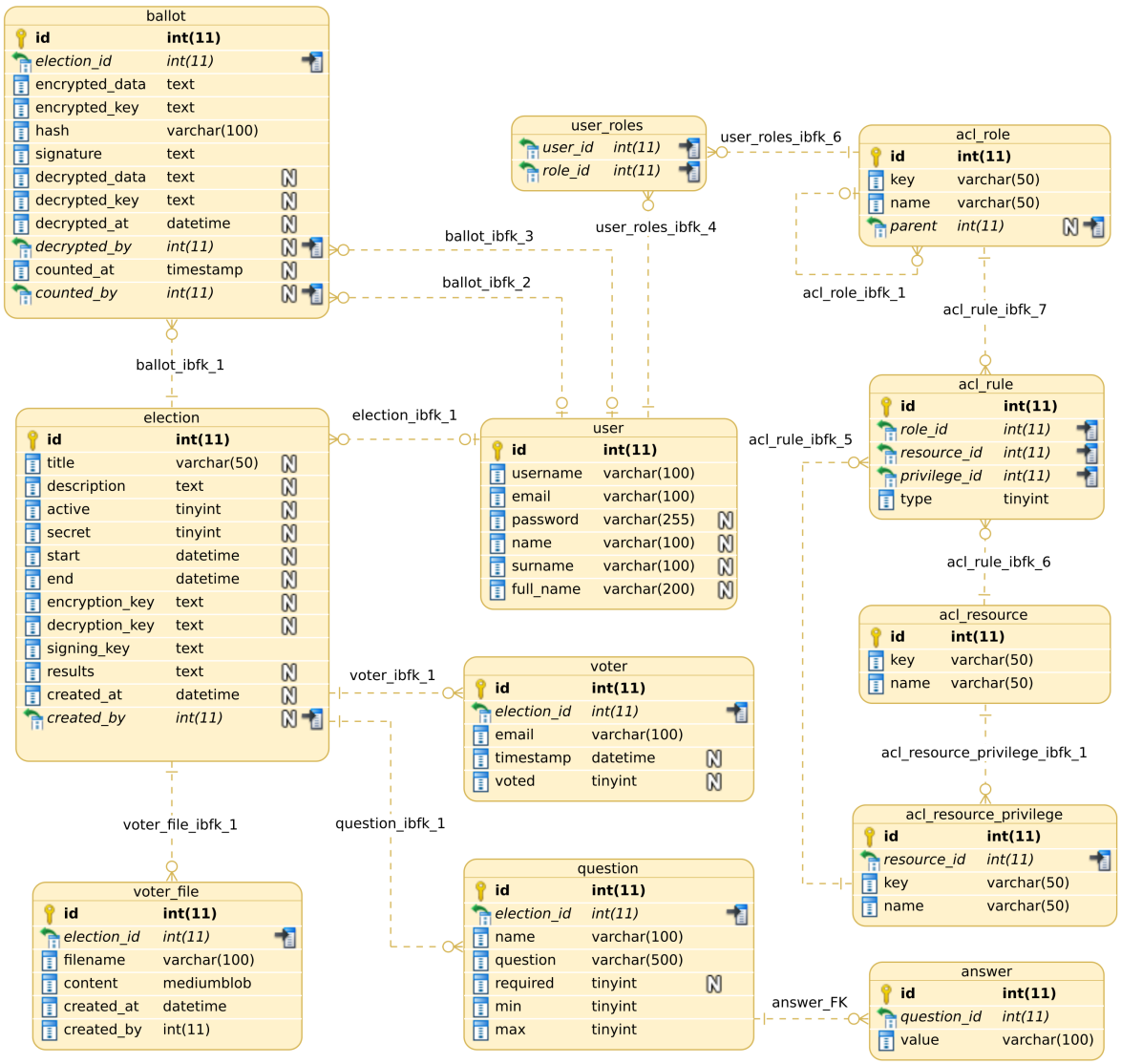
Obrázek 5.1 Diagram modelové vrstvy

Diagram na obrázku 5.1 obecně popisuje použité řešení, které bylo aplikováno na všechny entity. Třída `RoleDbMapper` implementuje rozhraní `RoleMapper`, které je předáno třídě `RoleRepository` jako závislost. V diagramu je použit příklad role, vyhledat ji lze podle Id nebo podle vazby na entitu `User`. Vhodné například, pokud chceme zjistit, které konkrétní role má uživatel nastaveny a tedy k jakým prostředkům a akcím má přístup díky pravidlům přiřazeným k jeho daným rolím. Zde je vhodné připomenout diagram na obrázku 3.3, který popisuje vazby mezi uživatelem, rolemi atd.

Metody `find()`, `findOne()`, `findAll()` a `findRelated()` slouží k vyhledání entit podle zadaných parametrů. Těmi může být filtr na Id nebo společnou vlastnost ale i vazba na jinou entitu. Metoda `getDataSource()` slouží k získání dat pro zobrazení interaktivních tabulek v administraci aplikace. Pomocí metody `create()` dokáže vytvořit nové instance entity `Role`, které jsou následně předány repozitáři samostatně (`findOne()`) nebo jako kolekce (některé mappery předávají objekt `...Collection`, některé předávají pole objektů).

5.2 Struktura databáze

Pomocí stanovených modelů je možné navrhnout podrobnější modely jednotlivých entit a tím pádem i strukturu databáze.

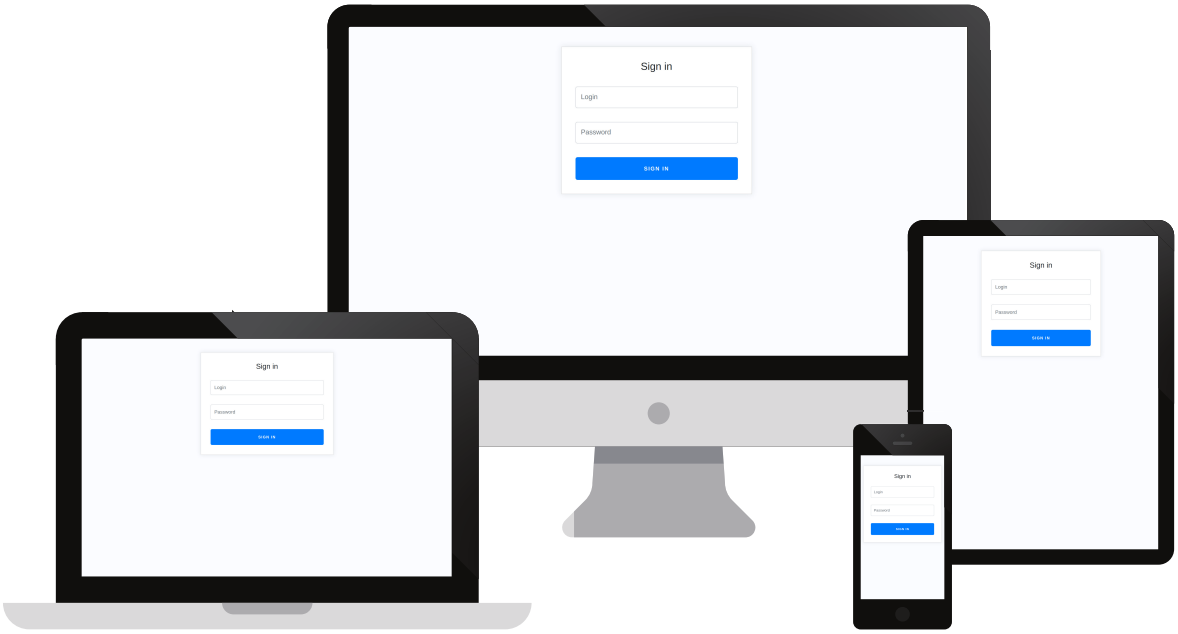


Obrázek 5.2 Entitně relační diagram

6 VRSTVA INFRASTRUKTURY

6.1 Přihlašovaní a autentizace

Přihlašování uživatelů probíhá identickým způsobem pro Frontend i Backend. Aplikace je dostupná pouze pro přihlášené, pokus o přístup bez platného přihlášení vyvolá přesměrování na SignPresenter. Přestože základní URL adresa aplikace vede na `Homepage:default`, (nepřihlášení) uživatelé jsou vždy přesměrováni nejdříve na přihlašovací stránku.



Obrázek 6.1 Vzhled přihlašovacího formuláře (zdroj: Freepik.com)

K ověření (autentizaci) uživatele pomocí kombinace uživatelského jména (e-mailové adresy) a hesla v Nette slouží rozhraní `Authenticator`[1]. Samotný proces ověření inicializuje objekt `User`[2]. SignPresenter po odeslání formuláře předá objektu `User` implementaci rozhraní a následně zavolá metodu `User::login($username, $password)`. Pokud ověření selže, je vyvolána výjimka `AuthenticationException`, která je zachycena v presenteru. Úspěšná autentizace způsobí uložení implementace `IIdentity` do objektu `User`.

Jedním z požadavků na aplikaci byla stanovena možnost přihlášení voličů pomocí univerzitních e-mailových adres. Implementace ověřování metodou Single Sign-On přes systém Shibboleth by byla nad rámec této práce a po diskuzi s vedoucím práce bylo zvoleno ověřování pomocí Active Directory přes LDAP. Dalším požadavkem byla možnost přihlášení externích uživatelů bez nutnosti vytvářet jim univerzitní e-mailové adresy. Tohoto bylo docíleno možností ověření vůči databázi aplikace.

[1] `Nette\Security\Authenticator`
[2] `Nette\Security\User`

```php
public function signInFormSuccess(Form $form, \stdClass $values): void
{
  $user = $this->getUser();
  try {
    $user->setAuthenticator($this->passwordAuthenticator);
    $user->login($values->username, $values->password);
    $this->redirect('Homepage:');
  } catch (AuthenticationException $ex) {
    try {
      $user->setAuthenticator($this->ldapAuthenticator);
      $user->login($values->username, $values->password);
      $this->redirect('Homepage:');
    } catch (AuthenticationException $ex) {
      $form->addError('Username or password invalid');
    } catch (NoConnectionException $ex) {
      $form->addError('LDAP server not available');
    }
  }
}
```

Fragment zdrojového kódu 6.1 Autentizace v SignPresenter

K autentizaci se využívají dvě implementace třídy `Authenticator`. První se k ověření použije `PasswordAuthenticator`, která při úspěšném ověření vrací entitu uživatele včetně všech rolí, při neúspěchu následuje pokus o ověření přes `LdapAuthenticator`.

- **PasswordAuthenticator** v prvním kroku získá na základě e-mailové adresy entitu uživatele z repozitáře. Pokud takový uživatel v databázi není, je vyvolána výjimka `AuthenticationException`. V druhém kroku předá uloženou hash hesla a ověřované heslo třídě `Nette\Security\Passwords` k porovnání. Pokud heslo neodpovídá, je opět vyvolána výjimka. Úspěšné ověření vrátí získanou entitu.
- **LdapAuthenticator** v prvním kroku se pokusí ověřit kombinaci e-mailové adresy a hesla vůči univerzitnímu Active Directory, při neúspěchu vyvolá výjimku. V druhém kroku se pokusí získat z repozitáře entitu uživatele s odpovídající e-mailovou adresou, pokud takový neexistuje, je vytvořena nová entita s rolemi získanými z Active Directory. Tyto role jsou buď *Student* nebo *Zaměstnanec*.

6.2 Oprávnění a autorizace

Ověření oprávnění uživatele provést akci (autorizace) na frontendové části je velice přímočaré. Zobrazit přehled aktivních voleb může kdokoli úspěšně autentizovaný systémem. Volit a zobrazit výsledky může kdokoli, kdo je uveden na seznamu voličů. Není třeba provádět žádnou dodatečnou autorizaci operací.

Backendová část je v tomto ohledu o něco složitější. Jednotliví uživatelé mohou mít přístup k presenteru, ale už ne k nějaké jeho konkrétní akci nebo signálu (metodám obecně). Podle modelu definovaného v části 3.4 byly vytvořeny jednotlivé třídy entit. Proces autorizace stejně jako autentizace inicializuje Nette objekt `User`[3]. V případě přihlašování uživatelů jsou mu předávány implementace napřímo, jelikož se používají dvě různé. U autorizace stačí předat jednu konkrétní implementaci, což je nejjednodušší provést v konfiguračním souboru aplikace. V soubrou `common.neon` tedy byly zaregistrovány služby `Permission`[4] a `AuthorizatorFactory`. Framework se o předání závislostí postará sám.

V rámci třídy `AuthorizatorFactory` jsou z repozitáře získány včechny prostředky a jejich akce a zaregistrovány v `Permission`. Dále se získají všechny role a postupně se zaregistrují společně s jejich pravidly. Jako poslední je zaregistrována role *superAdmin*, uživatel s touto rolí má nastaveno jediné pravidlo - vše povoleno.

```
1  public function create(): Authorizator
2  {
3    $authorizator = new Permission();
4    foreach ($this->resourceRepository->findAll() as $resource) {
5      $authorizator->addResource($resource->key, $resource->parent->key ?? null);
6    }
7    foreach ($this->roleRepository->findAll(true) as $role) {
8      $authorizator->addRole($role->key, $role->parent->key ?? null);
9      $rules = $role->rules->getByTypes();
10     foreach ($rules as $type => $ruleResources) {
11       foreach ($ruleResources as $ruleResource => $rulePrivileges) {
12         $authorizator->$type($role->key, $ruleResource, $rulePrivileges);
13       }
14     }
15   }
16
17   // allow all resources and privileges for superAdmin
18   $authorizator->allow('superAdmin');
19   return $authorizator;
20 }
```

Fragment zdrojového kódu 6.2 Tovární metoda třídy AuthorizatorFactory

[3] `Nette\Security\User`
[4] `Nette\Security\Permission`

Nette umožňuje pravidla nastavovat dvojím způsobem, výčtem povolených akcí a povolením všech akcí a výčtem akcí zakázaných. Aplikace umožňuje vytvořit pravidla obou typů, zakazující (deny) typ má nicméně smysl pouze pro roli superAdmin, která je jako jediná definována druhým způsobem. Ostatní role vždy obsahují pouze výčet povolených akcí, vše ostatní je zakázáno.

Zjistit, zda je uživatel oprávněn provést požadovanou akci, lze několika způsoby. Napřímo pomocí metody `User::isAllowed($resource, $privilege)`, která vrací `true`, pokud alespoň jedna z rolí uživatele k akci opravňuje, jinak vrací `false`. Tento způsob lze využít i v šablonách, kde je objekt uživatele automagicky [*sic*] dostupný jako proměnná `$user`. V presenterech se získá pomocí `$this->getUser()`a komponenty mají presenter dostupný jako členskou proměnnou, objekt `User` tedy získají pomocí `$this->presenter->getUser()`. Ostatním třídám aplikace se předává jako závislost v konstruktoru pomocí DI Containeru.

Jelikož presentery zpracovávají především požadavky od uživatele, podstatná část jejich metod by obsahovala opakující se volání metody `isAllowed`. Proto bylo využito anotací jednotlivých metod, příklad takové anotace je uveden ve fragmentu 6.3. Čtení anotací Nette usnadňuje pomocí objektu `MethodReflection`[5)], který je předáván metodě `checkRequirements()` každého presenteru. Tato metoda je v průběhu životního cyklu presenteru volána několikrát, poprvé při jeho vytvoření a předává se jí objekt `ComponentReflection`[6)] - reflexe aktuálního presenteru - a poté před každým *action*, *handle* a *render* v tomto pořadí s reflexí dané metody. Tímto bylo dosaženo velice efektivního zabezpeční jednotlivých částí presenteru.

```
1  /**
2   * @restricted
3   * @resource(elections)
4   * @privilege(delete)
5   */
6  public function handleDelete(int $id): void { ... }
```

Fragment zdrojového kódu 6.3 Příklad anotace metody

[5)] `Nette\Application\UI\MethodReflection`
[6)] `Nette\Application\UI\ComponentReflection`

Zpracování anotací probíhá v abstraktním presenteru `BasePresenter`. Pokud uživatel nedisponuje patřičným oprávněním, je mu zobrazena varovná zpráva (*flashMessage*) a je přesměrován na výchozí *View* aktuálního presenteru, pokud nemá oprávnění ani k tomu, je přesměrován na `HomepagePresenter`.

```
1  abstract class BasePresenter extends Nette\Application\UI\Presenter
2  {
3    public function checkRequirements($element): void
4    {
5      ...
6
7      if ($element instanceof Nette\Application\UI\MethodReflection &&
       ↪  $element->hasAnnotation('restricted')) {
8        $resource = $element->getAnnotation('resource');
9        $privilege = $element->getAnnotation('privilege');
10       if (!$user->isAllowed($resource, $privilege)) {
11         $this->flashMessage('You do not have permission to do that', 'warning');
12         if (!$user->isAllowed($resource, 'view')) {
13           $this->redirect('Homepage:');
14         }
15         if ($this->isAjax()) {
16           $this->forward('this');
17         } else {
18           $this->forward(':default');
19         }
20       }
21     }
22   }
23
24   ...
25 }
```

Fragment zdrojového kódu 6.4 Autorizace pomocí anotací metod

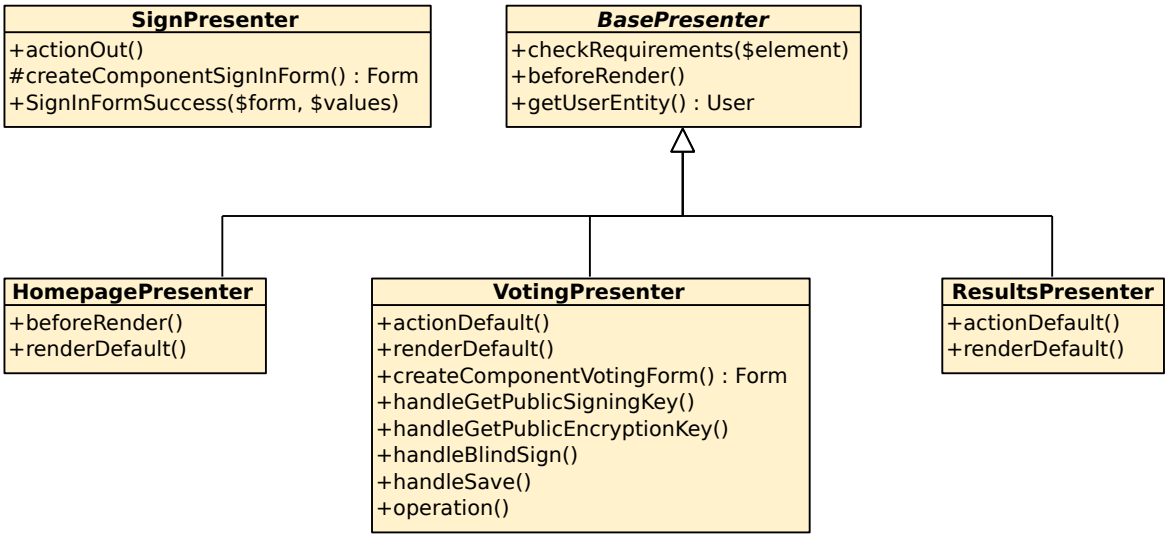
7 APLIKAČNÍ A PREZENTAČNÍ VRSTVA

Aplikační vrstva je tvořena především presentery, které obsluhují požadavky uživatele a pomocné třídy. Ty mohou být zcela samostatné nebo rozšiřují chování využívaných Composer balíčků. Prezentační vrstva je tvořena šablonami. Použitá architektura propojuje a částečně prolíná obě vrstvy, proto jsou uváděny společně. Toto prolínání podle Davida Grudla vidíme i v Nette, jak uvádí v komentáři u svého článku: "Dalo by se říct, že metoda `renderDefault()` je součástí vrstvy View, společně se šablonou." [https://zdrojak.cz/clanky/nette-framework-refactoring/?show=commentscomment-3554]

7.1 Presentery

V aplikaci se nachází tři skupiny presnterů, podle příslušnosti k modulu: Core, Backend a Frontend. Šablony pro konkrétní Presenter jsou ve společném adresáři `templates/jmenoPresenteru/` a název souboru každé šablony odpovídá akci daného presenteru. Pokud akce nevede na vykreslení šablony, nemusí mít šablonu vůbec vytvořenou. Příkladem je `Sign:out` (*SignPresenter* a akce *out*), která přihlášeného uživatele odhlásí a přesměruje na přihlašovací formulář `Sign:in` (stejný presenter, akce In). SignPresenter obsahuje metodu `renderIn` a tedy vyžaduje i šablonu `templates/Sign/in.latte`

7.1.1 Frontend



Obrázek 7.1 Třídy Presenter frontendové části

SignPresenter Jediným úkolem tohoto presenteru je uživateli poskytnout možnosti přihlášení a odhlášení.

Tabulka 7.1 Dostupné akce

akce	action*	render*	šablona
in	-	-	in.latte
out	actionOut	-	-

Dostupné akce:

- `Sign:in` Zobrazí přihlašovací formulář
  Nemá definované metody action ani render - ekvivalentní by byly prázdné metody. Jediným úkolem této metody je zobrazit formulář pro přihlášení, který se vytváří metodou `SignPresenter::createComponentSignInForm()`. Po odeslání formuláře se volá opět tato akce, nicméně se provede i callback formuláře `SignPresenter::signInFormSuccess()`. V tomto callbacku se aplikace pokusí o přihlášení uživatele pomocí poskytnutých údajů. Presenter je předá třídě implementující rozhraní `Nette\Security\IAuthenticator`. Princip přihlašování je popsán v části 6.1. Na tuto akci zároveň odkazují všichni potomci BasePresenter z metody `BasePresenter::checkRequirements()`, pokud se uživatel pokusí provést jakoukoli akci jako nepřihlášený (např. po vypršení session).
- `Sign:out` Odhlásí uživatele
  Uživatel po kliknutí na odkaz "Odhlásit" vyvolá akci tuto akci, kdy je odhlášen a následně přesměrován na přihlašovací formulář – `Sign:in` . Tato akce nemá render metodu ani šablonu (dochází k přesměrování, které předchází jakémukoli výstupu a ukončuje aktuální cyklus aplikace).

BasePresenter je abstraktní třída, která je společným předkem pro všechny presentery, které jsou dostupné pouze po přihlášení. Také obsahuje metody, které jsou užitečné pro všechny presentery obecně. Vzhledem k tomu, že se jedná o abstraktní presenter, nemá žádné metody action, render ani vlastní šablony. Metoda `checkRequirements()` slouží k ověření, že je uživatel přihlášen. Rozšiřuje rodičovskou metodu, která detekuje CSRF[1)] útoky, je tedy nezbytné zahrnout `parent::checkRequirements($element)`. Aby bylo zajištěno správné zobrazení krátkých stavových zpráv *flashMessage* a modálních oken během AJAXových požadavků, je zde i metoda `beforeRender()`, která se volá vždy před metodami `render`.

HomePagePresenter slouží jako rozcestník po přihlášení uživatele. Jeho jediná akce je `Homepage:default` , a obsahuje pouze render metodu, která získává objekty

[1)] Cross-Site Request Forgery

Election, které jsou dostupné danému uživateli.

VotingPresenter je stěžejním presenterem frontendové části aplikace, jejímž prostřednictvím probíhá celý proces volby na straně uživatele. Metoda `actionDefault()` získává a `renderDefault()` předává šabloně objekt `Election`. Samotný hlasovací lístek je tvořen formulářem. Ten je vytvářen tovární metodou `createComponentVotingForm()`. Zde bylo využito *generované továrničky*, což je zjednodušený zápis továrních tříd, které pouze vytváří jeden konkrétní objekt. Do presenteru je pomocí Dependency Injection předána závislost na rozhraní `VotingFormFactory` a Nette samo vygeneruje implementaci tohoto rozhraní, které je předáno do Presenteru [**?**].

Formulář `VotingForm` je samostatnou třídou rošiřující `Control`[2)] - v názvosloví Nette komponentou. Komponenty mohou mít vlastní šablony, což umžňuje zpřehlednit šablonu presenteru pokud není celý formulář vykreslován automaticky přes `FormRenderer`. Zároveň je i samotný kód presenteru jednodušší, o vytvoření formuláře, validaci a zpracování se totiž stará třída formuláře.

Hlasovací formulář neobsahuje žádnou logiku pro validaci ani zpracování odeslaných dat. Veškerá validace probíhá pomocí JavaScriptu na straně klienta a data se odesílají teprve po zašifrování a to přímo na presenter pomocí AJAX. Zpracování těchto požadavků (signálů) pomocí handle metod je popsáno v samostatné části 8 věnované zpracování hlasovacích lístků.

ResultsPresenter je velice jednoduchý presenter, který opět pomocí metod `actionDefault()` a `renderDefault()` získá a zpřístupní objekt `Election` šabloně k zobrazení výsledků hlasování / voleb.

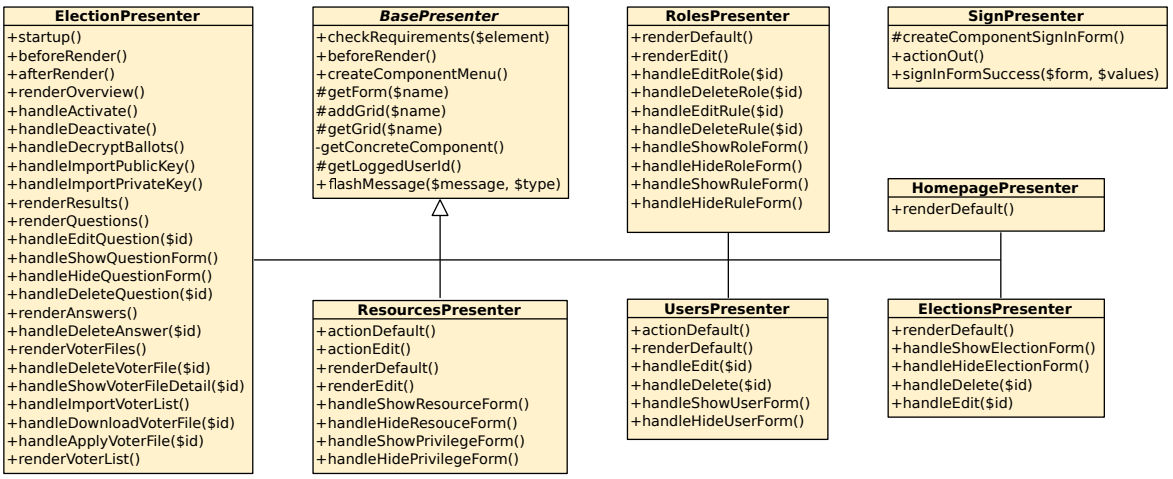
7.1.2 Backend

Tato část obsahuje nástroje pro správu uživatelů, uživatelských oprávnění a především voleb samotných. Je hojně využíváno formulářů a interaktivních tabulek (neboli datagridů). Formuláře a datagridy jsou samostatné komponenty[3)]. Jednoduché komponenty jsou zpravidla vytvářeny přímo v Presenteru. Složitější chování nebo obsáhlé komponenty je vhodnější oddělit do samostatné třídy. Podrobněji jsou datagridy popsány v části 7.3.1.

Datagridy jsou vytvářeny pomocí třídy `App\Backend\Utils\DataGrid\Datagrid`, která zefektivňuje způsob vytváření jednotlivých gridů díky zapouzdření často

[2)] `Nette\Application\UI\Control`
[3)] třídy rozšiřující `Nette\Application\UI\Control`

používaných konstrukcí. Tím se nejen výsledný kód v presenterech zjednoduší, ale zároveň gridy napříč aplikací mají stejné chování a na všechny se aplikuje stejná logika. Tímto přístupem se také podstatně usnadní změny ovlivňující všechny gridy - například změna ikony se projeví všude.



Obrázek 7.2 Třídy Presenter backendové části

SignPresenter slouží ke správě přihlášení do aplikace, v zásadě se jedná o obdobu frontendového `SignPresenter`. Popis této třídy je tedy identický.

BasePresenter je stejně jako `App\Frontend\Presenters\BasePresenter` abstraktní třídou společnou pro všechny další presentery. Ověřuje, že uživatel je přihlášen a má oprávnění provést požadovanou akci.

HomepagePresenter hlavní jeho funkcí je umožnit zobrazení navigace po přihlášení i pro uživatele, který nemá definovaná žádná práva a jediné co může v aplikaci provést je se odhlásit. Mohl by zobrazovat i nějakou formu rozcestníku, jako ve frontendové části, to ale obstarává navigační lišta. Dalším vhodným využitím tohoto presenteru by byla prezentace důležitých dat formou *dashboardu*.

ElectionsPresenter poskytuje přehled nad všemi vypsanými volbami, jejich rychlou editaci, mazání a odkaz na zobrazení detailu. Nové volby se také vytváří v tomto presenteru. Při vypisování nových voleb je vhodné zvolit výstižný krátký popisek, jeho editace je uživatelsky přívětivá díky JavaScriptovému pluginu *tinyMCE*.

ElectionPresenter nabízí detailní přehled konkrétních voleb rozdělený do záložek. Některé záložky se zobrazují pouze pokud se volby nacházejí v určitém stavu. Záložka výsledky se například zobrazí až po ukončení voleb.

V záhlaví detailu se nachází kontextové menu, které umožňuje volby (de)aktivovat, nahrávat seznamy voličů a klíče volební komise. Aktivace voleb způsobí jejich zobrazení voličům, hlasování je umožněno až v řádném termínu. Po aktivaci voleb není možné měnit jejich nastavení, ale lze je opět deaktivovat. Aktivovat a deaktivovat volby lze pouze pokud se nenacházejí v průběhu jejich konání. Po skončení voleb se voličům ukazují pouze výsledky a to do doby než jsou deaktivovány.

Jednotlivými záložkami jsou:

- **Overview** , kde se zobrazuje formátovaný popis voleb a tři pole s šifrovacími klíči (pokud jsou dostupné). Těmito klíči jsou: privátní a veřejný klíč volební komise a veřejný podpisový klíč serveru.
- **Results** s výsledky voleb. Tato záložka je distupná pouze po ukončení voleb. V případě, že dosud nejsou spočítané výsledky, je nabídnuto jejich spočítání a následně se již zobrazují pouze výsledky v přehledných grafech.
- **Questions** zobrazí grid otázek definovaných pro dané volby. Otázkou může být volená pozice, její odpověďmi pak jména kandidátů. Při založení nové otázky je možné nastavit minimální a maximální počet otázek, zda je odpověď povinná a jednotlivé odpovědi. V gridu je možné otázky editovat a mazat.
- **Answers** obsahuje grid odpovědí na všechny otázky, zde je možné odpovědi mazat.
- **Voter list** zobrazí grid s aktivním seznamem voličů.
- **Voter files** v tomto gridu se nachází všechny soubory se seznamem voličů, které byly pro dané volby nahrány. Soubory je možno prohlížet, mazat, stáhnout a aplikovat vybraný soubor jako aktivní seznam voličů. Nový soubor lze nahrát přes kontextové menu v záhlaví.

UsersPresenter jednoduchý presenter s gridem uživatelů a formulářem pro přidávání / editaci uživatelů. Formulář umožňuje uživatelům přidělovat role i změnit heslo. Aplikace neobsahuje žádný registrační formulář, nové uživatele musí vždy přidávat osoba s patřičným oprávněním - pravděpodobně administrátor aplikace. Jak bylo popsáno v části 6.1, není potřeba zakládat uživatelské účty voličům, ale pouze osobám, kterým je potřeba navýšit oprávnění.

RolesPresenter výchozí View tohoto presenteru nabízí grid všech dostupných rolí a formulář pro jejich základní editaci. Pomocí akce gridu je možné zobrazit detail role,

kde se nachází seznam definovaných pravidel přístupu ve formě gridu. Tato pravidla lze editovat, mazat a přidávat nová.

ResourcesPresenter velice podobý předchozímu RolePresenter. Tento umožňuje správu prostředků (*Resource*), na stránce detailu je pak k dispozici správa akcí (*Privilege*) dostupných pro daný prostředek.

7.1.3 Core

Tento modul je společný pro frontendovou i backendovou část a obsahuje pouze uživatelsky přívětivé zpracování chybových stavů. Pokud se uživatel pokouší přistoupit na neexistující stránku, není nalezen požadovaný záznam v databázi (HTTP 404) nebo nemá uživatel potřebná oprávnění k zobrazení stránky (HTTP 403), místo základních chybových stránek HTTP serveru mu je zobrazena chybová stránka vygenerovaná Nette. Stejně tak v případě chyby serveru (HTTP 500).

V případě, že se Nette nachází ve vývojovém režimu, jsou všechny chyby aplikace předávány ke zpracování nástroji Tracy (dříve Laděnka) [**?**], který vypíše chybu včetně části zdrojového kódu, předávaných proměnných, dotazů na databázi a dalších velice užitečných informací pro ladění chyb. V produkčním režimu jsou chyby předávány ErrorPresenteru, který chyby 4xx předává dále do Error4xxPresenter případně rovnou předá statickou šablonu s chybou 500 nebo 503.

7.2 Šablony

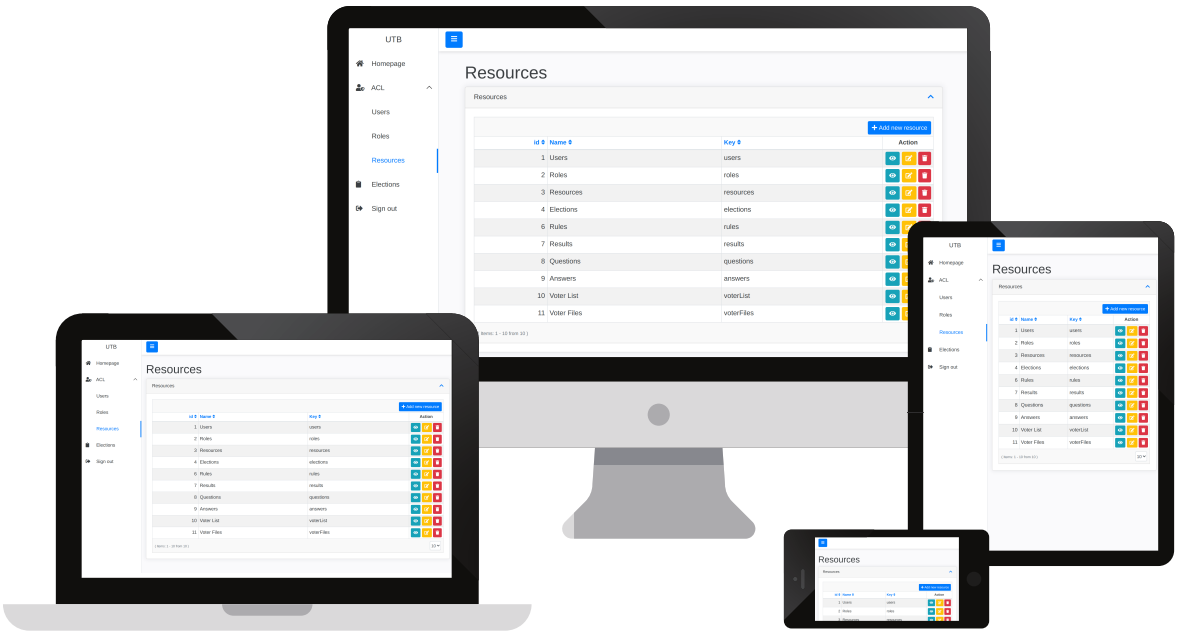
7.2.1 Skripty a balíčky

7.2.2 CSS styly

7.3 Pomocné třídy

7.3.1 Datagridy

Zkráceně gridy, tyto interaktivní tabulky umožňují kromě zobrazení dat i jejich filtraci, stránkování, akce nad řádkem tabulky (např. odkaz směřující na Signál) a mnoho dalších funkcí. Gridy přijímají data jako objekt `Datasource`, který může obsahovat data ve formě obyčejného pole nebo dotazu SQL [**?**]. Kromě jednoho případu v této aplikaci vždy pracují s SQL dotazem. Výhoda tohoto přístupu je minimalizování potřebného objemu dat k zobrazení gridu. Po změně filtrace nebo stránky je vždy upraven SQL dotaz a až následně odeslán na databázový server, filtrace a stránkování tedy probíhá přímo na SQL serveru, nikoli pomocí PHP nebo JavaScriptu.



Obrázek 7.3 Vzhled datagridu (zdroj: Freepik.com)

Akce nad řádkem jsou typicky úlohy typu editace, mazání zobrazení detailu a směřují na Signál presenteru (metody handle). Jsou reprezentovány ikonami. Požadavek na vymazání řádku (záznamu) je navíc opatřen potvrzovacím dialogovým oknem, aby nedošlo k nechtěnému smazání při nechtěném kliknutí.

8 ZPRACOVÁNÍ HLASOVACÍCH LÍSTKŮ

Nejcitlivější částí aplikace je bez pochyb právě práce s hlasovacími lístky, proto jí byla věnována samostatná část. Hlasovací lístek je na straně klienta (voliče) reprezentován formulářem. Tento formulář je generován v Nette na základě specifikací nastavených pro dané volby. Každý hlasovací lístek musí obsahovat nejméně jednu otázku. Otázka obsahuje několik odpovědí a limit pro nejmenší a největší povolený počet zvolených odpovědí. Otázka může být také označena jako povinná.

Z výše uvedeného vyplývá, že formulář je potřeba validovat - zjistit, že odpovědi uvedené na hlasovacím lístku odpovídají specifikacím voleb. Nette umožňuje pravidla validace nastavit již při vytváření formuláře a po jeho odeslání klientem je formulář zvalidován na straně serveru. Neúspěšná validace přeruší zpracování formuláře a klientovi poskytne zpětnou vazbu (chybové zprávy). Pokud je načtena JavaScriptová knihovna `netteForms.js`, provádí se validace navíc i na straně klienta [1].

Pokud by byl hlasovací lístek odeslán serveru k validaci, byla by narušena anonymita voleb. A to i v případě, že by server jen ověřil, že počet odpovědí odpovídá nastaveným limitům. Data by server měl k dispozici nešifrovaná a stejně tak identitu voliče, kdokoli by mohl vznést oprávněnou námitku, že takto není zaručena absolutní anonymita voliče. Validace formuláře z tohoto důvodu probíha pouze na straně klienta - v prohlížeči.

Ve fragmentu 8.1 jsou vidět skripty a balíčky, které jsou využívány pro validaci a šifrování hlasovacích lístků na straně klienta. Třída `Crypto` obsahuje metody pro šifrování dat a komunikaci se serverem za účelem výměny klíčů, z tohoto důvodu jsou ji předávány odkazy na signály presenteru.

```
1  <script src="/js/crypto/jsbn@latest.js"></script>
2  <script src="/js/crypto/js-sha256@latest.js"></script>
3  <script type="module">
4    naja.registerExtension(new ValidateVotingForm());
5    import Crypto from '/js/crypto.js'
6    window.crypt = new Crypto({
7      publicEncryptionKeyLink: {link getPublicEncryptionKey!},
8      publicSigningKeyLink: {link getPublicSigningKey!},
9      signingLink: {link blindSign!},
10     savingLink: {link save!}
11   });
12 </script>
```

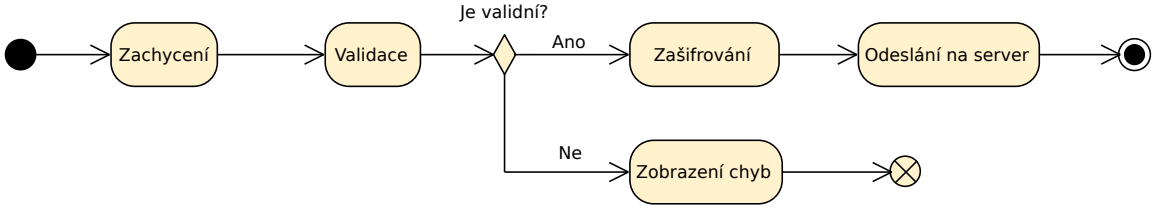
Fragment zdrojového kódu 8.1 JavaScript použitý při hlasování

8.1 Validace

Validace formuláře na straně klienta je všeobecně považována za pouhé usnadnění pro uživatele, umožňuje rychle a přehledně uživatele informovat o nesrovnalostech ve formuláři. Nicméně, veškerá data přijatá od klienta (prohlížeče) by měla být považována za potenciálně nebezpečná a nevalidní, jelikož upravit data před odesláním nebo upravit JavaScriptové validační skripty není nemožné. [2][3][4]

Je nutné tedy počítat s tím, že data jsou validována již na straně klienta, ale pouze pro potřeby voliče - aby mu byly přehledně komunikovány jakékoli chyby, kterých se během vyplňování hlasovacího lístku dopustil. Na server se data dostanou zašifrována a nebylo by již možné ověřit jejich platnost. To by mohlo snadno vyústit ve vysoké procento neplatných hlasů a v horším případě četným námitkám proti platnosti voleb samotných. Nutnost validovat data na straně serveru nicméně zůstává, pouze se odkládá na dobu, kdy budou data serverem čitelná - sčítání hlasů.

Po vyplnění formuláře a kliknutí na potvrzovací tlačítko následuje série dílčích kroků, které vedou k odeslání zašifrovaného hlasovacího lístku nebo prezentaci chybových hlášení.



Obrázek 8.1 Diagram aktivity validace

Formulář má nastaveno odesílání pomocí AJAX, událost odeslání formuláře je tedy nejprve zachycena knihovnou Naja. V šabloně je do Naja registrováno rozšíření `ValidateVotingForm`, což je jedno z vlastních rozšíření zahrnutých v soubrou `Naja.ext.js`.

Všechna potřebná pravidla pro validaci jsou již nastavena v Nette při generování formuláře a klient má načtenou knihovnu pro validaci formulářů od Nette, k validaci je tedy použita tato knihovna. Bohužel prezentace chyb touto knihovnou není uživatelsky nejpřívětivější, bylo tedy zvoleno validování jednotlivých elementů formuláře samostatně. Pomocí `HTMLSelectElement.setCustomValidity()` je nevalidním prvkům změněn stav validity, který využívá framework Bootstrap k zobrazení validovaného formuláře (pomocí pseudoelementů `:valid` a `:invalid`). V případě, že formulář obsahuje nevalidní prvky, je zobrazena chybová zpráva, zvýrazněny chybné elementy a proces ukončen.

```
validateForm(event) {
  $('form input[type=submit]').attr('disabled', true)
  const { element, originalEvent } = event.detail;
  Nette.formErrors = [];
  for (let el of element.form.getElementsByTagName('input')) {
    if (el.dataset.netteRules !== undefined) {
      el.setCustomValidity(Nette.validateControl(el) ? '' : 'invalid')
    }
  }

  if (originalEvent) {
    originalEvent.stopImmediatePropagation();
    originalEvent.preventDefault();
  }
  event.preventDefault();

  if (Nette.formErrors.length) {
    $(element.form).addClass('was-validated')
    this.showErrors();
    toastr.error('There were errors in the form')
    $('form input[type=submit]').attr('disabled', false)
    return;
  }

  ...
}
```

Fragment zdrojového kódu 8.2 část třídy ValidateVotingForm

8.2 Šifrování

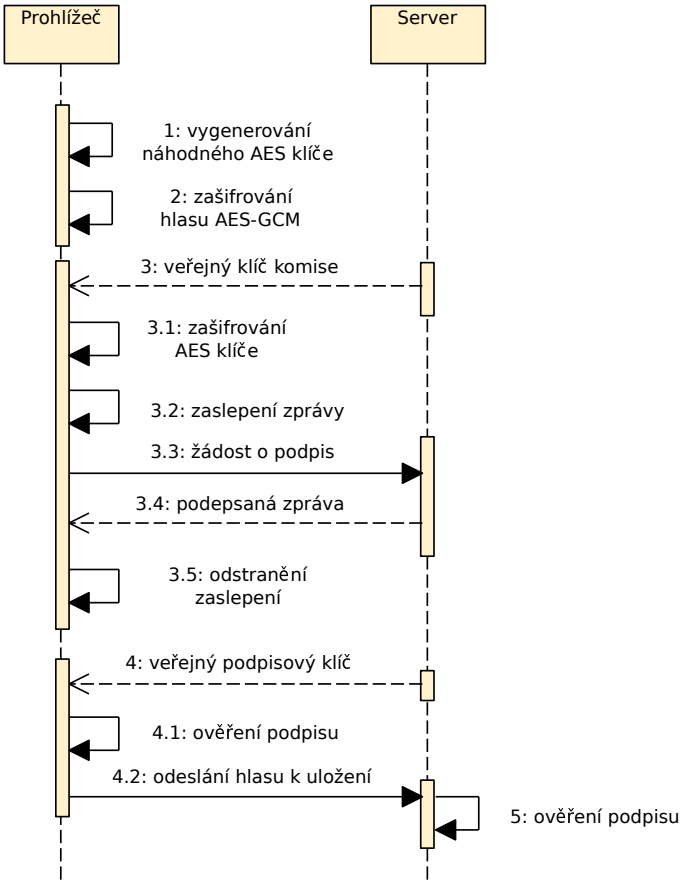
Data zvalidovaného formuláře jsou uspořádána do asociativního pole a předána k zašifrování třídě `Crypto`. Tato třída má ze šablony předány odkazy pro komunikaci se serverem a již po svém instanciování si od serveru vyžádá veřejný klíč volební komise a veřejný podpisový klíč serveru (oba RSA) a vygeneruje nový náhodný klíč AES-GCM (256-bit) a *nonce* (inicializační vektor). Jednotlivé operace jsou prováděny asynchronně na pozadí, uživatele tedy nijak neomezují.

Ve chvíli, kdy je hlas předán třídě Crypto je volební lístek (pole) převeden na JSON řetězec a zašifrován pomocí vygenerovaného AES klíče. Díky použité metodě šifrování je výsledný šifrovaný text pro stejný otevřený text pokaždé jiný. Následně je zašifrován i samotný AES klíč a to RSA klíčem volební komise (s RSA-OAEP výplní). Dále je na sha-256 hash šifrovaného klíče aplikován náhodný faktor zaslepení r podle algoritmu popsaného v části [odkaz na kapitolu]. Zaslepená zpráva je odeslána na server k podpisu.

Server přijímá zaslepenou zprávu, která neobsahuje žádné informace o hlasovacím lístku, jedná se pouze o hash klíče použitého k zašifrování hlasovacího lístku. Aby

bylo možné na straně klienta z podepsané zprávy odstranit náhodný faktor r, není možné použít žádné výplně. Použitá knihovna `phpseclib` navíc jako dodatečnou ochranu podpisového klíče proti časovým útokům používá zaslepování[5], které by rovněž znemožnilo odstranění faktoru r, proto nebylo použito. Podepsaná zpráva je vrácena zpět klientovi k dalšímu zpracování.

Na straně klienta je z podepsané zprávy odstraněn náhodný faktor r reverzní operací zaslepení. Tím je získán validní podpis původní (nezaslepené) zprávy - zašifrovaného AES klíče. Pomocí veřejného podpisového klíče serveru a hashe zašifrovaného AES klíče je ověřena správnost podpisu. Pokud podpis zprávy odpovídá originálu, na server se odešle k uložení zašifrovaný hlasovací lístek, zašifrovaný AES klíč a podpis.



Obrázek 8.2 Sekvenční diagram šifrování

8.3 Ukládání

V tuto chvíli se nabízí dotaz proč aplikovat slepé podepisování, když server, který vystavuje podpis zároveň zpracovává originál zprávy, kterou podepsal. Tento způsob byl zvolen s ohledem na univerzálnost použití. Právě díky slepému podepisování může být k ukládání hlasů použit jiný server zcela nezávislý na zbytku systému. Tento nezávislý server si pomocí veřejného podpisového klíče ověří, že zprává (hlasovací lístek) odpovídá

podpisu a je důvěryhodná = podepsaná volebním serverem. Vzhledem k tomu, že se podepisuje pouze hash, je možné připojit i samotná data hlasovacího lístku, nezávislý server by tedy přijímal pouze zprávu a její podpis. Není to nicméně nutností, protože zašifrovaná data mohou být dešifrována pouze jedním klíčem a ten je ověřený volebním serverem.

Po přijetí dat k uložení tedy server opět ověří že přijatý zašifrovaný AES klíč odpovídá podpisu, který ho provází. Následně vytvoří entitu `EncryptedBallot`, nastaví ji přijatá data a předá repozitáři k uložení. Zároveň je u voliče zaznamenán čas, kdy hlasoval, tím pádem mu není umožněno hlasovat ve stejných volbách znovu. Jak bylo řečeno výše, ověření a uložení přijatých dat může provést i jakákoli důvěryhodná třetí strana, volební server pouze potřebuje vědět, že volič úspěšně hlasoval.

8.4 Sčítání

Sčítání hlasů je umožněno pouze po skončení voleb a to uživatelům s patřičným oprávněním. Před sečtením hlasů je nutné je nejdříve dešifrovat, za tímto účelem musí člen volební komise serveru zpřístupnit privátní RSA klíč odpovídající klíči, který byl nahrán volební komisí před začátkem voleb. Z důvodu co nejvyšší možné důvěryhodnosti volebního systému bylo zvoleno řešení, kdy server (nebo kdokoli s přístupem k němu) nemá možnost hlasy dešifrovat dříve než volební komise uvolní potřebný RSA klíč. Tento způsob ovšem předpokládá správné vygenerování a bezpečné uložení klíče volební komisí.

Při nahrávání veřejné části RSA klíče je kontrolováno jestli se jedná o veřejný RSA klíč, který aplikace dokáže použít. Zodpovědnost za poskytnutí správného klíče je ovšem na volební komisi. Pokud by byl poskytnut například klíč, jež je chráněn heslem, aplikace by v současné verzi hlasy nedokázala dešifrovat. Zároveň není umožněno vyzkoušet kompatibilitu veřejného a privátního klíče s aplikací, aby server nepřišel do kontaktu s privátním kíčem dříve než je to absolutně nezbytné.

Pro volby s ukončeným hlasováním je v backendové části aplikace zpřístupněna volba nahrání privátního klíče a v záložce výsledků také tlačítko pro sečtení hlasů. Kliknutí na tlačítko vede na signál `countBallots`. Proces sčítání hlasů řídí třída `BallotCounter`, která je závislostí `ElectionPresenter`. Jediná veřejná metoda této třídy je `processBallots($election): array`, která deleguje dešifrování a validaci hlasovacích lístků na třídy `BallotDecryptor` a `BallotValidator` a samotné sečtení hlasů provádí sama. Jednotlivé metody této třídy jsou vidět ve fragmentu 8.3.

```php
public function processBallots(Election $election): array
{
  $this->election = $election;
  $decrypted = $this->ballotDecryptor->setElection($election)->decryptBallots();
  [$valid, $invalid, $errors] =
  ↪  $this->ballotValidator->setElection($election)->validateBallots();
    $this->counter = [
      'valid' => count($valid),
      'invalid' => count($invalid),
      'error' => count($errors),
    ];
    $this->prepareCounter();
  $this->countResults($valid);
  return $this->counter;
}

private function prepareCounter()
{
  foreach ($this->election->getQuestions() as $question) {
    $this->counter[$question->getId()] =
      array_fill_keys(array_keys($question->getAnswers()->getIdValuePairs()),
      ↪  0);
  }
}

/**
 * @var DecryptedBallot[] $ballots
 */
private function countResults(iterable $ballots)
{
  try {
    foreach ($ballots as $ballot) {
      $data = $ballot->unpackData();
      foreach ($data['questions'] as $questionId => $answers) {
        foreach ($answers as $answerId => $value) {
          $this->counter[$questionId][$answerId]++;
        }
      }
      $ballot->setCountedAt(new \DateTime())
        ->setCountedBy(UserId::fromValue($this->user->getId()));
      $this->ballotRepository->save($ballot);
    }
  } catch (\JsonException $e) {
    $this->log($e, Logger::CRITICAL, $ballot);
  } catch (SavingErrorException $e) {
    $this->log($e, Logger::WARNING, $ballot);
  }
}
```

Fragment zdrojového kódu 8.3 Metody třídy BallotCounter

Třída `BallotDecryptor` obsahuje dvě veřejné metody volané z třídy `BallotCounter`, metoda `setElection()` pouze nastaví aktuálně zpracovávané volby a `decryptBallots()` nejprve načte zašifrované hlasovací lístky z repozitáře, následně metodou `verify($ballot)`ověří, že hash a podpis uložené v databázi odpovídá zpracovávanému volebnímu lístku a nakonec dešifruje AES klíč, kterým dešifruje samotná data. Dešifrovaná data jsou poté nastavena novému objektu `DecryptedBallot`, který je repozitářem uložen. V případě chyby při ověřování, dešifrování a dalších, je chyba zaznamenána do logu (souboru) včetně identifikace hlasovacího lístku. Kód dešifrování je vidět na fragmentu 8.4.

Třída `BallotValidator` má za úkol ověřit, že všechny dešifrované hlasovací lístky jsou platné a splňují pravidla nastavená pro dané volby. V zásadě se jedná o pozdní validaci hlasovacího formuláře, jak bylo popsáno v části 8.1. Tato třída obsahuje identické metody jako `BallotDecryptor` pro nastavení voleb a zpracování lístků. Z repozitáře jsou získány všechny dešifrované volební lístky a postupně ověřena data v nich obsažená, že splňují nastavená kritéria. Jmenovitě, že identifikátor voleb souhlasí s aktuálně zpracovávanými a že povinné otázky jsou zodpovězeny a všechny otázky splňují limity pro minimální a maximální počet odpovědí. Kód validace je vidět na fragmentech 8.5 a 8.6.

8.5 Výsledky

Výstupem validace jsou tři pole volebních lístků - platné, neplatné a chybné. Neplatné jsou ty, které neprošly validací (nesprávný počet odpovědí apod.), chybné jsou pak ty, které se nepovedlo zpracovat. Všechny chyby jsou opět zaznamenány do logu. Třída `BallotCounter` si připraví počítadlo pro všechny možné odpovědi v daných volbách a platné lístky jsou předány ke sčítání metodě `countResults($ballots)`. Ta projde každou otázku a navýší počítadlo pro každou vyplněnou odpověď. Konečný stav počítadla je pak vrácen do `ElectionPresenter`, který nastaví výsledky do objektu `Election` a předá ho repozitáři k uložení. Od té chvíle jsou dostupné výsledky voleb k zobrazení v obou částech aplikace. Na frontendové části se zobrazí pouze oprávněným voličům a to pouze pokud jsou dané volby stále aktivní. Po deaktivaci voleb se výsledky zobrazují pouze v backendové části. Výsledky voleb je rovněž možné stáhnout do počítače ve formě PDF protokolu z kontextové nabídky v detailu voleb.

```php
/**
 * @throws DecryptionException
 */
private function decryptKey(string $encryptedKey): DecryptingKey
{
  static $decryptingKey;
  if ($decryptingKey === null) {
    $decryptingKey = $this->election->getPrivateEncryptionKey();
    if ($decryptingKey === null) {
      throw new InvalidStateException('Decrypting key has not been set yet.');
    }
  }

  try {
    $decrypted = $decryptingKey->decrypt(base64_decode($encryptedKey));
    ['key' => $key, 'iv' => $iv] = json_decode($decrypted, true, 512,
    ↪  JSON_THROW_ON_ERROR);
    return new DecryptingKey($key, $iv);
  } catch (\Exception $e) {
    throw new DecryptionException('Decrypting AES key failed');
  }
}

/**
 * @throws VerificationException
 */
private function verify(EncryptedBallot $ballot): void
{
  $hash = $this->hash($ballot->encryptedKey);
  if (bin2hex($hash) !== $ballot->hash) {
    throw new VerificationException('hash does not match the stored value');
  }
  $signature = $this->sign($hash);
  if (bin2hex($signature) !== $ballot->signature) {
    throw new VerificationException('signature does not match the stored
    ↪  value');
  }
}

private function hash(string $message): string
{
  static $sha;
  if ($sha === null) {
    $sha = new Hash('sha256');
  }
  return $sha->hash($message);
}

private function sign(string $message): string
{
  static $key;
  if ($key === null) {
    $key = $this->election->getPrivateSigningKey();
  }
  return $key->decrypt($message);
}
```

Fragment zdrojového kódu 8.4 Metody třídy BallotDecryptor

```php
/** @param DecryptedBallot[] $ballots */
private function validate(array $ballots): array
{
  $valid = $invalid = $error = [];
  foreach ($ballots as $ballot) {
    try {
      $data = $ballot->unpackData();
      $this->checkElection((int) $data['electionId']);
      $this->checkQuestions($data['questions']);
      $valid[] = $ballot;
    } catch (\JsonException $e) {
      $error[] = $ballot;
      $this->log($e, Logger::CRITICAL, $ballot);
    } catch (ValidationException $e) {
      $invalid[] = $ballot;
      $this->log($e, Logger::WARNING, $ballot);
    }
  }
  return [$valid, $invalid, $error];
}

/** @return Question[] */
private function getQuestions(): array
{
  static $questions;
  if ($questions === null) {
    foreach ($this->election->getQuestions() as $question) {
      $questions[$question->getId()] = $question;
    }
  }
  return $questions;
}

private function getAnswers(int $questionId): array
{
  static $answers;
  if ($answers === null) {
    foreach ($this->getQuestions() as $qId => $question) {
      $answers[$qId] = $question->getAnswers()->getIdValuePairs();
    }
  }
  return $answers[$questionId];
}
```

Fragment zdrojového kódu 8.5 Metody třídy BallotValidator (1)

```
44
45  /**
46   * @throws ValidationException
47   */
48  private function checkElection(int $value): void
49  {
50    static $electionId;
51    if ($electionId === null) {
52      $electionId = $this->election->getId();
53    }
54    if ($electionId !== $value) {
55      throw new ValidationException('Wrong election id');
56    }
57  }
58
59  /**
60   * @throws ValidationException
61   */
62  private function checkQuestions(array $tested): void
63  {
64    foreach ($this->getQuestions() as $qId => $question) {
65      if (empty($tested[$qId]) && !$question->required) {
66        continue;
67      }
68      if ($question->required && empty($tested[$qId])) {
69        throw new ValidationException('missing required question');
70      }
71      $answerCount = count($tested[$qId]);
72      if (($question->getMin() > $answerCount) || ($question->getMax() <
       ↪  $answerCount)) {
73        throw new ValidationException('Wrong number of answers');
74      }
75      $this->checkAnswers($qId, $tested[$qId]);
76
77    }
78  }
79
80  /**
81   * @throws ValidationException
82   */
83  private function checkAnswers(int $questionId, array $tested): void
84  {
85    $answers = $this->getAnswers($questionId);
86    foreach ($tested as $key => $value) {
87      if (!array_key_exists($key, $answers) || $answers[$key] !== $value) {
88        throw new ValidationException('answer does not match');
89      }
90    }
91  }
```

Fragment zdrojového kódu 8.6 Metody třídy BallotValidator (2)

ZÁVĚR

Text závěru.

SEZNAM POUŽITÉ LITERATURY

[1] HAŠKA, D.: *Porovnání PHP frameworků pro tvorbu internetové aplikace.* Bakalářská práce, Vysoká škola ekonomická v Praze, Praha, 2016.

[2] FOWLER, M.: *Patterns of enterprise application architecture.* Boston: Addison-Wesley, 2003, ISBN 0-321-12742-0.

[3] Zdroják.cz: Prezentační vzory z rodiny MVC [online]. Dostupné z: https://zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/, 2009, [cit. 2021-05-07].

[4] FOWLER, M.: Martin Fowler: Passive View [online]. Dostupné z: https://www.martinfowler.com/eaaDev/PassiveScreen.html, 2006, [cit. 2021-05-07].

[5] Docs, N. .: Presentery [online]. Dostupné z: https://doc.nette.org/cs/3.1/presenters#toc-action-action, 2021, [cit. 2021-05-07].

[6] Docs, N. .: Routování [online]. Dostupné z: https://doc.nette.org/cs/3.1/routing, 2021, [cit. 2021-05-07].

[7] EVANS, E.: *Domain-driven Design: Tackling Complexity in the Heart of Software.* Boston: Addison-Wesley, 2004, ISBN 0-321-12521-5.

[8] MARTIN, R. C.: *Agile Software Development, Principles, Patterns, and Practices.* New York: Pearson, první vydání, 2002, ISBN 978-0135974445.

[9] Nette: Best practice: formuláře jako komponenty [online]. Dostupné z: https://pla.nette.org/cs/best-practice-formulare-jako-komponenty#toc-ui-control, 2021, [cit. 2021-05-07].

[10] Tracy: ladicí nástroj se kterým je radost chybovat [online]. Dostupné z: https://tracy.nette.org/, 2021, [cit. 2021-05-07].

[11] Contributte Datagrid: DataGrid for Nette framework [online]. Dostupné z: https://contributte.org/packages/contributte/datagrid/, 2021, [cit. 2021-05-07].

[12] Docs, N. .: Validace formulářů [online]. Dostupné z: https://doc.nette.org/cs/3.1/form-validation, 2021, [cit. 2021-05-07].

[13] Truth, S.: Do Not Rely On Client-Side Validation [online]. Dostupné z: https://blog.securityinnovation.com/blog/2011/07/do-not-rely-on-client-side-validation.html, 2011, [cit. 2021-05-07].

[14] Exchange, I. S. S.: Is HTML5 input pattern validation sufficient (or even relevant) for client-side validation? [online]. Dostupné z: https://security.stackexchange.com/questions/169771/is-html5-input-pattern-validation-sufficient-or-even-relevant-for-client-side, 2017, [cit. 2021-05-07].

[15] Jovanovic, J.: Web Form Validation: Best Practices and Tutorials [online]. Dostupné z: https://blog.securityinnovation.com/blog/2011/07/do-not-rely-on-client-side-validation.html, 2009, [cit. 2021-05-07].

[16] Wigginton, J.: Phpseclib API Documentation: Crypt_RSA [online]. Dostupné z: https://api.phpseclib.com/1.0/Crypt_RSA.html, 2021, [cit. 2021-05-07].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MVC	Model View Controller
MVP	Model View Presenter
DDD	Domain Driven Design
ACL	Access Control List
SRP	Single Responsibility Principle
CSRF	Cross-site request forgery
LDAP	Lightweight Directory Access Protocol

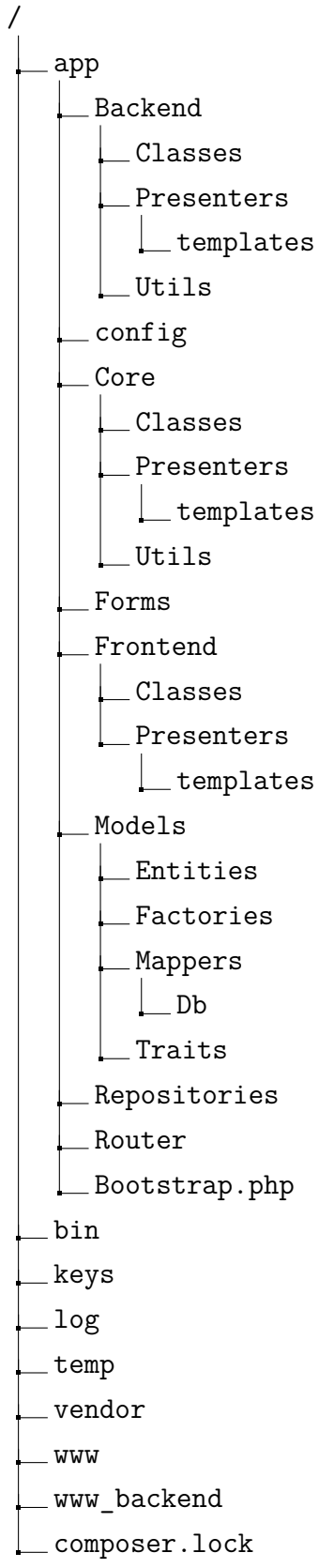
SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM FRAGMENTŮ ZDROJOVÉHO KÓDU

SEZNAM PŘÍLOH

PŘÍLOHA P I. ADRESÁŘOVÁ STRUKTURA APLIKACE



PŘÍLOHA P II. BALÍČKY TŘETÍCH STRAN

2.1 Datagrid