Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a implementace webové a mobilní aplikace pro vodoměry

Tomáš Holý

Bakalářská práce
2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš Holý**
Osobní číslo:	**I15077**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Návrh a implementace webové a mobilní aplikace pro vodoměry**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je návrh a implementace webové aplikace pro správu vodoměrů a dále mobilní aplikace na platformě Android pro jednoznačnou identifikaci vodoměru pomocí čárového kódu a zadání odečtu. Součástí zpracování teoretických východisek práce je popis klíčových technologií použitých při tvorbě aplikací s důrazem na jazyk PHP, framework Laravel a dále popis alternativních technologií, kterými může být výstup práce také zpracován včetně základního srovnání popsaných technologií. V praktické části se student zaměří na vlastní návrh, implementaci a popis navrhovaných aplikací. Výstupem práce bude webová aplikace pro správu vodoměru na straně dodavatele a základní přehled pro zákazníka zobrazující průběh spotřeby vody, dalším výstupem bude mobilní aplikace pro načtení stavu vodoměru zákazníkem.

Rozsah pracovní zprávy:	**min. 30 stran**
Rozsah grafických prací:	
Forma zpracování bakalářské práce:	**tištěná**

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 7. 2020

Tomáš Holý

PODĚKOVÁNÍ

Mé poděkování patří Ing. Monice Borkovcové, Ph.D. za rady při zpracování bakalářské práce. Děkuji také mé rodině a přátelům za podporu po dobu studia.

ANOTACE

Tato bakalářská práce se zabývá vývojem webové aplikace pro správu vodoměru a mobilní aplikace určené pro zadání odečtu vodoměru. Práce popisuje klíčové technologie použité při tvorbě aplikací, popis alternativních technologií včetně základního srovnání popsaných technologií s důrazem na PHP framework Laravel a jazyk Java. Součástí je návrh a implementace aplikací a popis základních aspektů navrhovaných aplikací a požadované technologie. Webová aplikace slouží pro správu vodoměru vodárenskou společností s možností zobrazení základních informací pro zákazníka včetně spotřeby jednotlivých odběrných míst. Mobilní aplikace slouží pro jednoznačnou identifikaci vodoměru pomocí čísla vodoměru a zadání nového odečtu zákazníkem. Uživatel mobilní aplikace přihlašovacích údajů webové aplikace a může spravovat vytvořené importy odečtů prostřednictvím REST API webové aplikace.

KLÍČOVÁ SLOVA

Webová aplikace, mobilní aplikace, Laravel framework, Android, prohlížeč, databáze, vodoměr, odečet, spotřeba vody.

TITLE

Design and Implementation of Web and Mobile applications for Water Meter

ANNOTATION

This bachelor thesis is focused primarily on the development of a web application focused on water meter management and a mobile application used for entering new water consumption allowance. The thesis describes key technologies used for production of the apps, alternative technologies including foundational comparison of similar techniques used for development with emphasis on PHP framework Laravel and Java language. The Design and implementation parts are realized by describing essential aspects of development and required technologies. The web app serves as a managing tool for water meters of customers for distributors with and as a basic overview of water consumption on customer's side. The mobile app allows customers to unambiguously identify a water meter and enter respective water consumption allowances. Mobile app user needs to be identified by web app login and can manage past allowance imports though web app's REST API.

KEYWORDS

Web application, mobile application, Laravel framework, Android, browser, database, water meter, allowance, water consumption.

OBSAH

Seznam obrázků **8**
Seznam tabulek **9**
Seznam zkratek **10**
Úvod **11**
Teoretická část **12**
1 Framework Laravel **12**
1.1 Historický vývoj 13
1.2 Instalace 13
1.3 Adresářová struktura 14
1.4 Konfigurace 16
1.5 Základní vlastnosti 17
1.5.1 Artisan 17
1.5.2 MVC 17
1.5.3 Model 19
1.5.4 Route, Request a Middleware 20
1.5.5 Controller 21
1.5.6 View a šablony Blade 21
1.5.7 Migrace 22
1.5.8 Eloquent 23
1.5.9 Frontend 24
2 Alternativní PHP frameworky **25**
2.1 Symfony 25
2.2 CakePHP 25
2.3 CodeIgniter 25
2.4 Zend 26
2.5 Srovnání frameworků 26
3 Základní technologie pro práci s frameworkem Laravel a související **27**
3.1 Composer 27
3.2 HTML 28
3.3 PHP 29
3.4 CSS 30
3.5 JavaScript 31
3.6 Java 33
3.7 Databáze 33
3.8 MySQL 35
3.9 Php Storm 35
Praktická část **36**

4 Aplikační řešení.....36
4.1 Technologie použité při vývoji.....36
4.1.1 API.....36
4.1.2 Android Studio.....37
4.1.3 Postman.....38
4.2 Požadavky.....38
4.2.1 Funkční požadavky.....39
4.2.2 Nefunkční požadavky.....40
4.3 Návrh databáze.....41
4.3.1 Tabulky.....42
4.4 Diagram aktivit.....44
4.5 Diagram případů užití.....45
4.6 Shrnutí návrhu aplikací.....46
4.7 Implementace webové aplikace.....46
4.7.1 Grafický design.....46
4.7.2 Uživatelské role.....46
4.7.3 Přihlašovací modul.....47
4.7.4 Navigace v aplikaci.....48
4.7.5 Moje odečty.....48
4.7.6 Přehled spotřeby.....49
4.7.7 Administrátorská část.....51
4.7.8 Tvorba REST API.....52
4.8 Implementace mobilní aplikace.....53
4.8.1 Grafický design.....53
4.8.2 Připojení k REST API webové aplikace.....54
4.8.3 Autentifikace uživatele.....54
4.8.4 Jednoznačná identifikace vodoměru.....55
4.8.5 Navigace v aplikaci.....56
4.8.6 Prohlížení a nahlášení spotřeby.....57
Závěr.....58
Použitá literatura.....59
Přílohy.....61

SEZNAM OBRÁZKŮ

Obrázek 1: Logo frameworku Laravel....12
Obrázek 2: Konfigurační soubor config/database.php 16
Obrázek 3: Diagram MVC....18
Obrázek 4: Relace modelu....19
Obrázek 5: RESTful akce pro uživatele ve web.php odkazující na příslušný controller 20
Obrázek 6: Migrace tabulky uživatelů aplikace 22
Obrázek 7: Databázový model....41
Obrázek 8: UML Activity diagram....44
Obrázek 9: UML Use case diagram....45
Obrázek 10: Přihlašovací obrazovka webové aplikace....47
Obrázek 11: Uvítací obrazovka 48
Obrázek 12: Modul Moje odečty 49
Obrázek 13: Modul Odběrná místa....50
Obrázek 14: Modul Uživatelské odečty....51
Obrázek 15: Postman - test API....53
Obrázek 16: Přihlášení do mobilní aplikace…….55
Obrázek 17: Výběr čísla vodoměru…….. 55
Obrázek 18: Hlavní menu…...……...……56
Obrázek 19: Přehled vytvořených odečtů…….56
Obrázek 20: Nový uživatelský odečet……...57
Obrázek 21: Editace uživatelského odečtu……57
Obrázek 22: Přehled spotřeby……...57

SEZNAM TABULEK

Tabulka 1: Srovnání frameworků……. 26
Tabulka 2: Funkční požadavky webové aplikace……. 39
Tabulka 3: Funkční požadavky mobilní aplikace……..40
Tabulka 4: Nefunkční požadavky webové aplikace……..40
Tabulka 5: Nefunkční požadavky mobilní aplikace…….....
.....411

SEZNAM ZKRATEK

MVC	Model-View-Controller
REST	Representational State Transfer
JSON	JavaScript Object Notation
API	Application Programming Interface
PHP	Hypertext Preprocessor
ORM	Object Relational Mapping
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheet
HTML	Hypertext Markup Language
MySQL	My Structured Query Language
RDBMS	Relational Database Management System
SQL	Structured Query Language
DDL	Data Definition Language
DML	Data Manipulation Language
DCL	Data Control Language
DQL	Data Query Language
TCL	Transaction Control Language

ÚVOD

Se stále rostoucí poptávkou po nových technologiích, které může dnes využívat běžný uživatel i s nejlevnějšími chytrými telefony, i vodárenské společnosti poptávají technologie, kterými si mohou získat nové zákazníky. Spojením s dalšími technologiemi, jako je jednoznačná rádiová identifikace vodoměrů, či unikátní čárové kódy, vznikají nové způsoby, jak získat náskok před konkurencí. Jedním z těchto způsobů je nabídnout svým zákazníkům možnost zapsat vlastní odečet.

Cílem této práce je návrh a implementace dvou aplikací pro prohlížení historie odečtů odběrných, míst včetně grafického přehledu, s možností zapsání nového odečtu na straně zákazníka a správy vodoměrů, odběrných míst a jejich odečtů, včetně zápisu odečtů do informačního systému vodárenské společnosti. Webová aplikace, která je hlavním projektem této práce, umožňuje výše zmíněné akce a zároveň slouží jako přístupový bod REST API, který uživatelům mobilní aplikace umožňuje vzájemnou komunikaci mezi aplikacemi. Mobilní aplikace na platformě Android umožňuje pomocí přihlašovacích údajů webové aplikace a unikátního přístupového tokenu komunikaci s REST API a po jednoznačné identifikaci vodoměru umožní uživateli vytvořit nový odečet, prohlížet historii nezapsaných odečtů a zobrazit grafický přehled spotřeby.

Teoretická část práce je zaměřena na popis technologií použitých při vývoji webové aplikace s důrazem na PHP frameworku Laravel. Druhá kapitola je věnována popisu a porovnání alternativních PHP frameworků, kterými mohla být výsledná aplikace vytvořena. Poslední kapitola teoretické části je věnována popisu základních technologií pro práci s frameworkem.

V praktické části, se autor zaměřuje na představení a popis programů a technologií použitých při vývoji výsledných aplikací. Následující kapitoly se soustředí na návrh aplikací, které obsahují funkční a nefunkční požadavky reálné vodárenské společnosti. Součástí konceptu jsou UML diagramy aktivit a případů užití a relační model databáze včetně popisu tabulek. Obsahem posledních kapitol praktické části je popis implementace těchto požadavků v návazných aplikacích. V závěru práce dojde k jejich zhodnocení a zamyšlení se nad dalším vývojem těchto aplikací. Výsledné aplikace jsou cílené pro vodárenské společnosti, které chtějí poskytnout možnost nahlášení vlastního odečtu a prohlížení historie spotřeby svým zákazníkům a správu těchto odečtů ze strany vodárenské společnosti.

TEORETICKÁ ČÁST

1 FRAMEWORK LARAVEL

Když vznikaly první dynamické webové aplikace, vývojáři museli psát nejen unikátní logiku, ale i komponenty, které jsou pro většinu aplikací společné – uživatelská autentifikace, validace dat, připojení k databázi, a další. V dnešní době existují desítky frameworků a tisíce komponent a knihoven, které jsou veřejně dostupné. S příchodem těchto technologií zodpovědnost za vývoj a údržbu izolovaných částí kódu, které mají jasnou funkcionalitu, přešla na vývojáře, kteří této konkrétní problematice rozumí lépe. Framework tvoří soubor kolekcí komponent a knihoven třetích stran, které jsou spojeny do jednoho řešení pomocí konfiguračních souborů, poskytovatelů služeb, předepsaných adresářových struktur a samozaváděcích programů. Základní výhodou využití frameworku před psaním čistého PHP kódu je použití již existujících řešení tak, jak mají být správně používány společně vedoucím k mnohem rychlejšímu a efektivnějšímu vývoji. S použitím frameworku jsou dána jasná pravidla, jakým způsobem se má programovat tak, aby jeden vývojář mohl začít přesně tam, kde druhý přestal a aby přesně věděl, kde a co hledat. Dodržováním těchto pravidel lze docílit snadnému porozumění kódu jiného vývojáře a efektivně navázat vlastním kódem. Ty nejlepší frameworky nabízí jednak solidní základ pro kompletní vývoj aplikací, ale nabízí i svobodu v upravování těchto komponent tak, jak vývojář uzná za vhodné. [1]

Obrázek 1: Logo frameworku Laravel (zdroj KIMČÍK. Laravel 6 je tady! [online]. [cit. 28.6..2020]. Dostupný na WWW: https://laravelblog.cz/clanek/laravel-6-je-tady)

Následující kapitoly představí stručnou historii frameworku Laravel, instalaci a základní konfiguraci, adresářovou strukturu projektu, dále základní vlastnosti, jejichž znalost je základem pro správný vývoj ve frameworku a budou představeny bezpečnostní techniky využívané Laravelem a základní techniky práce s databázovými systémy.

1.1 Historický vývoj

Jedním z prvních frameworků pro vývoj webových aplikací byl Ruby on Rails z roku 2004. Těžko lze nalézt framework, který nebyl Ruby nějakým způsobem ovlivněn. Díky Ruby došlo k popularizaci MVC, RESTful JSON API, jednotné konfigurace a mnoho dalších nástrojů a jasně definovaných pravidel, které ovlivnily způsob rychlého vývoje aplikačního vývoje. Roku 2005 vznikl CakePHP, následovaly Symfony, CodeIngiter, Zend framework a Kohana. Některé z těchto frameworků se snažily napodobit Ruby, soustředíc se na databázové objektově relační mapování (ORM), MVC architektury, a další nástroje pro rychlý vývoj. Jiné, mezi které se řadí Symfony a Zend framework, se zaměřily na komerční řešení a návrhové šablony pro podniky. Kolem roku 2010 byl nejpopulárnější CodeIgniter, který stavěl na jednoduchosti použití, výborné dokumentaci a velké komunitě vývojářů. Časem začal zaostávat za konkurencí, především z důvodu představení nových technologií, například příchod PHP 5.3 a novinek jako použití jmenných prostorů, verzovacích systému Git a také programu Composer.

Proto se pan Taylor Otwell rozhodl vytvořit framework Laravel, který vyšel v první beta verzi v červnu roku 2011. Do roka přišly další dvě verze, konkrétně Laravel 2 a 3. V květnu 2014 vyšel kompletně přepsaný Laravel 4 postavený na komponentech frameworku Symfony a sadě vlastních komponent jménem Illuminate, stahovaných pomocí programu Composer. Jelikož Laravel závisí na komponentech Symfony, adaptoval půlroční plán uvolňování nových verzí po vzoru Symfony. Leden 2015 přinesl novou verzi Laravelu s pořadovým číslem 5. Následující verze přinesly především opravy chyb a bezpečnostních hrozeb. Nejnovější verze je v současnosti 7.16.1. [1]

1.2 Instalace

Základním požadavkem pro běh jakéhokoliv PHP frameworku je samozřejmě instalace PHP, a to ve verzi vyhovující minimálním požadavkům frameworku. Laravel má několik stabilních verzí a je doporučeno nový projekt vždy stavět na nejnovější. Autorem vyvinutá webová aplikace je postavena na verzi 7.17.2 a zde budou uvedeny požadavky pro tuto verzi.

Následuje výčet minimálních požadavků, které musí být instalovány na serveru nebo počítači, kde je aplikace vyvíjena:

- PHP >= 7.2.5
- BCMath PHP Extension

- Ctype PHP Extension
- Fileinfo PHP extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Globální instalace programu pro správu závislostí Composer je také nutností. Pokud jsou výše uvedené požadavky splněny, lze pomocí příkazové řádky zadat příkaz `composer create-project laravel/laravel nazev_projektu` a v nově vytvořené podsložce s názvem *nazev_projektu* je vytvořen nový projekt. [1]

1.3 Adresářová struktura

Následuje výčet složek a souborů tvořících kostru nově vytvořeného projektu včetně základního popisu:

app/ - V této složce jsou umístěny modely, controllery, notifikace, service providery a mnoho dalšího. Jádro aplikačního kódu, logika, výjimky, rozhraní, vše má své místo v jedné z podsložek.

bootstrap/ - Obsahuje soubory, které Laravel spouští při startu.

config/ - Zde jsou umístěny konfigurační soubory.

database/ - Migrace, seedery, továrny pro generování testovacích dat jsou umístěny v této složce.

public/ - Veřejná složka, kam server odkazuje při spuštění aplikace. Jsou zde umístěny uložené soubory, fotky, skripty, kaskádové styly a index.php, který nastartuje procesy pro routování a requesty.

resources/ - Soubory, které jsou potřebné pro další skripty, pohledy, lokalizační soubory a zdroje JavaScript a CSS souborů mají své místo zde.

routes/ - Všechny definice routování aplikace, konzole a Artisan příkazů jsou umístěny v této složce.

storage/ - V této složce jsou umístěny aplikační logy, cache a kompilované systémové soubory a skripty.

tests/ - Jednotkové testy a integrační testy patří sem.

vendor/ - Do této složky program Composer instaluje své závislosti. Je ignorována Gitem, protože je očekáváno, že při každém umístění projektu bude nově generována Composerem.

.editorconfig - Soubor obsahuje instrukce pro vývojové prostředí nebo textový editor, jak má pracovat s projektem, například délka odsazení, charset, ořezávání bílých znaků apod.

.env a *.env.example* - Hlavní konfigurační soubor projektu, obsahuje proměnné, které budou jiné na každém serveru, kde je aplikace umístěna.

.gitattributes a *.gitignore* - Konfigurační soubory pro Git.

artisan - Umožňuje spuštění Artisan příkazů z příkazového řádku.

composer.json a *composer.lock* - Konfigurační soubory pro Composer. Composer.json je uživatelsky editovatelný, přidávají se zde nové závislosti projektu. Composer.lock obsahuje aktuální verze závislostí, aby vývoj na více počítačích probíhal vždy na stejných verzích.

package.json - Podobný jako composer.json, akorát pro frontendové položky a závislosti. Slouží jako instrukce pro NPM (více v kapitole 3.5)

phpunit.xml - Konfigurační soubor pro jednotkové testování PHPUnit, které Laravel standardně používá.

server.php - Záložní server pro méně výkonné servery, které chtějí zobrazit aplikaci.

webpack.mix.js - Konfigurační (nepovinný) soubor pro Laravel Mix. Zřetězením jsou zpracovány soubory pro CSS a JavaScript. Ve standardní odkazuje, kam zkompilovat CSS a JavaScript (do složky *public/*) [1]

1.4 Konfigurace

Hlavní konfigurace frameworku Laravel, nastavení připojení k databázi, mailování, a další, se nachází ve složce *config/*. Každý soubor vrací pole hodnot, ke kterým lze přistupovat kdekoliv v aplikaci pomocí klíčových slov oddělených tečkami. Například příkaz pro získání jména databáze je `config('database.connections.mysql.username')`. [1]

config > database.php

You, a few seconds ago | 2 authors (qwasyx0 and others)

```php
<?php       qwasyx0, 3 months ago • Added full project

use Illuminate\Support\Str;

return [

    'default' => env('DB_CONNECTION', 'mysql'),

    'connections' => [

        'mysql' => [
            'driver' => 'mysql',
            'url' => env('DATABASE_URL'),
            'host' => env('DB_HOST', '127.0.0.1'),
            'port' => env('DB_PORT', '3306'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'unix_socket' => env('DB_SOCKET', ''),
            'charset' => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
            'prefix' => '',
            'prefix_indexes' => true,
            'strict' => true,
            'engine' => null,
            'options' => extension_loaded('pdo_mysql') ? array_filter([
                PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
            ]) : [],
        ],
```

Obrázek 2: Konfigurační soubor config/database.php (zdroj vlastní)

Proměnné, které jsou rozdílné pro každý server, například právě přístupové údaje k databázi, se získávají ze souboru *.env*, kde se nachází řádek *DB_USERNAME = root*, ke kterému lze přistoupit pomocí `env('DB_USERNAME', 'vychozi_hodnota')`. Stejným způsobem, tedy pomocí teček, lze přistoupit k souborům umístěných například ve složkách *routes/* a *storage/*. [1]

1.5 Základní vlastnosti

V následujících kapitolách jsou charakterizovány základní vlastnosti Laravelu, díky kterým je nejoblíbenějším PHP frameworkem. Funkčnosti, které by zabraly dlouhé hodiny vývoje a generovaly desítky souborů lze pomocí příkazového řádku vytvořit a poté upravovat dle konvencí frameworku tak, aby byl zajištěn rychlý a bezproblémový vývoj aplikace.

1.5.1 Artisan

Artisan je základní komponenta Laravelu, postavená na Symfony komponentě Console, která umožňuje tvorbu příkazů pro příkazový řádek. Pomocí *php artisan* příkazů se Laravelu sděluje, jakou komponentu chce vývojář vytvořit, migrovat databázi, vyčistit cache, hostovat server, nebo kompletně přenastavit frontend, a mnoho dalších. Tento způsob manipulace se soubory je preferovaný, vývojář se tak vyvaruje chybám, jako třeba špatné umístění souboru, překlepům apod. Velmi užitečným nástrojem přístupným v příkazové konzoli je Tinker (příkaz `php artisan tinker`), který umožňuje výpis databázových tabulek pomocí modelů, jejich plnění novými daty pomocí továren, přístup k relacím pomocí ORM a ukládání nových hodnot. [1]

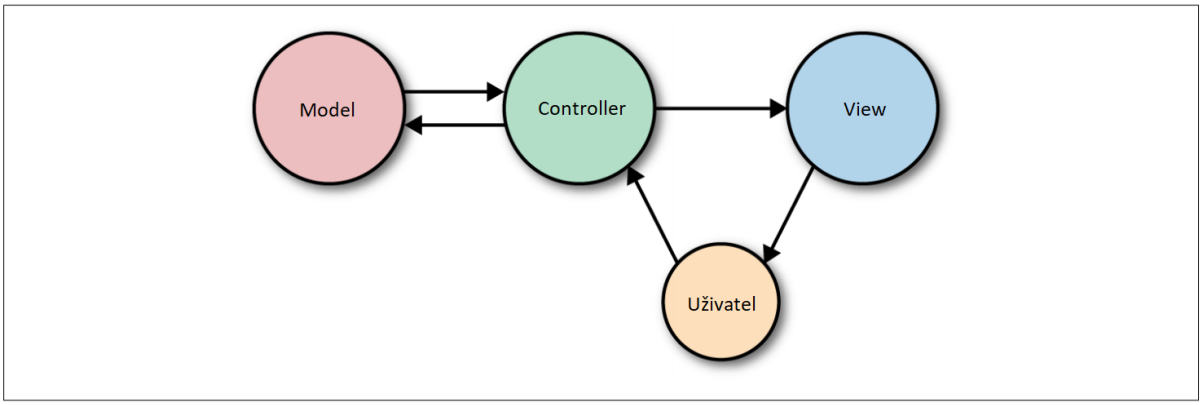
Nástroj také umožňuje definici vlastních příkazů pomocí `php artisan make:command jmeno_prikazu`. Tímto příkazem je generován nový soubor ve složce *app/Console/Commands/jmeno_prikazu.php*. S nově vytvořeným souborem se pracuje jako s třídou rozšiřující rozhraní Command, lze definovat podpis, popis, konstruktor a samotné tělo příkazu pomocí metody handle(). [1]

1.5.2 MVC

Návrhový vzor MVC vznikl v 70. letech 20. století společně s vývojem prvních grafických rozhraní. Byl jedním z prvních pokusů o popsání návrhu softwarových konstrukcí z hlediska rozdělení zodpovědnosti. V dnešní době MVC popisuje architekturu, která rozděluje programovou logiku do tří propojených, ale zároveň nezávislých komponent, tak, že modifikace jedné má minimální vliv na ostatní. Tyto komponenty jsou datový model (Model), pohled (View) a controller (Controller). Počáteční písmena těchto komponent dávají dohromady název architektury, jako je to i s obdobnými návrhovými vzory MVP a MVVM. [3]

Zde je nutné podotknout, že následující podrobnější popis a příklad implementace architektury je specifický pro PHP framework Laravel, který je hlavním tématem teoretické části. U jiných frameworků se může interpretace lišit, především v rozdělení zodpovědností jednotlivých komponent.

Datový model v aplikaci spravuje data a reprezentuje tabulku databáze. Řadič obsahuje logiku aplikace, pracuje s modely dle requestů zaslanými uživatelem aplikace a zobrazí požadovaná data pomocí pohledu. Následující diagram ilustruje vzájemnou vazbu těchto komponent. [1]



Obrázek 3: Diagram MVC (zdroj [1])

Pro pochopení základní funkcionality bude uveden jednoduchý příklad. Uživatel zadá adresu webové aplikace a je mu zobrazena přihlašovací stránka aplikace, na které jsou umístěna pole pro zadání přihlašovacích údajů. Toto je pohled, kam uživatel zadává data a potvrdí je stisknutím tlačítka. Tímto zašle HTTP request metodou POST obsahující přihlašovací údaje do řadiče. Řadič provede kontrolu zaslaných dat a může uživateli vrátit např. pohled se zprávou, že heslo není dostatečně dlouhé. Pokud projde request validací, je zavolán model, který v tomto případě reprezentuje tabulku, kde jsou uloženi uživatelé aplikace. Řadič pomocí modelu získá data z tabulky a ověří, zda takové přihlašovací údaje existují. Pokud jsou zadané údaje správné, je uživateli zaslán pohled jeho domovské stránky. V opačném případě je uživateli zaslán pohled s opětovným přihlášením a se zprávou, že uživatel neexistuje.

V současné době je koncept MVC užíván především pro návrh webových aplikací. Zajišťuje flexibilitu a spolehlivost při rozšiřování o novou funkcionalitu díky oddělení aplikační a prezentační logiky. Existuje mnoho frameworků využívajících tuto architekturu, jejich základnímu popisu a srovnání se věnuji v kapitole 2. [1] [3]

1.5.3 Model

Modely reprezentují entity databázového modelu. Jsou umístěny ve složce app\ a dle konvencí se pojmenovávají stejně jako entity, tedy jednotným číslem, na rozdíl od tabulek reprezentovaných tabulek databáze, které se pojmenovávají množným číslem. Vytváří se pomocí příkazu `php artisan make:model Nazev`. V modelu lze určit, jakou tabulku má reprezentovat pomocí atributu *protected $table = 'nazev_tabulky';*. Manipulace s modelem probíhá přes controller, Laravel sám ví, s jakým záznamem se manipuluje, a k určitému atributu se přistupuje pomocí klíčového slova this, tedy pro zjištění id záznamu lze zadat *$this->id;*. V modelu lze také definovat nový atribut, který v tabulce není, např. vypočítaná spotřeba, pomocí funkce *public function getSpotrebaAttribute() { return 1; };*, který je nutno přidat k modelu pomocí *protected $appends = ['spotreba'];*. Tento je pak dostupný stejně jako ostatní atributy pomocí *$this->spotreba;*. [1]

Relace mezi jednotlivými modely se nastavují pomocí veřejných funkcí. Příkladem může být odběrné místo, které má své odečty a umístěný vodoměr. Definováním relací lze poté přistoupit k jednotlivým atributům pomocí zřetězení, například po vložení následujícího kódu lze zjistit číslo vodoměru pomocí *$this->vodomer->cislovodomeru;*.

```
28      public function odecetvodomeru()
29      {
30          return $this->hasMany(Odecetvodomeru::class)->orderBy('novy_stav','desc');
31      }
32
33      public function vodomer()
34      {
35          return $this->hasOne(Vodomer::class);
36      }
37
```

Obrázek 4: Relace modelu (zdroj vlastní)

Oproti tomu relace M:1, což je výše zmíněný odečet, by nefungovala, protože není jenom jeden a Laravel místo určitého modelu vrátí tzv. kolekci. V této kolekci se musí vybrat určitý záznam, např. první odečet odběrného lze získat zadáním *$this->odecetvodomeru->first()->novy_stav;*. Co jsou to kolekce a jak se s nimi pracuje, bude vysvětleno v kapitole 1.5.8. Jelikož jsou odečty seřazeny sestupně dle nového stavu, Eloquent vrátí odečet s největším novým stavem. Použitím těchto relací se vývojář vyhne dlouhým SQL dotazům a přepisováním souvisejících SQL dotazů při přidání nového atributu v tabulce. Laravel se o vše postará sám. Aplikace se tak stává snadno rozšiřitelnou s narůstajícím počtem tabulek. [2]

1.5.4 Route, Request a Middleware

Routování je koncept nastavení nové URI adresy zadávané do internetového prohlížeče. Umožňuje pohyb v aplikaci pomocí HTTP protokolu (využívaném většinou internetových prohlížečů) na novou stránku, získávání nových dat zobrazovaných uživateli, nebo zobrazení chybových stavových kódů HTTP, jako je třeba známá chyba 404. Všechny routy jsou umístěné ve složce *routes/,* která v nové instalaci obsahuje čtyři soubory, jež umožní vývojáři definovat konzolové příkazy, vysílací kanály, API přístupové body a navigaci v aplikaci. Nejdůležitějším souborem je *web.php*, bez kterého by aplikace vůbec nemohla fungovat. Definují se v něm adresy zadávané do internetového prohlížeče a odkazují na určitý controller, případně metodu definovanou v controlleru, kde je umístěný kód definující, co bude daná routa dělat. [2]

```
27
28  Route::get('uzivatel','UsersController@index')->name('user.index')->middleware('auth');
29  Route::get('uzivatel/create','UsersController@create')->name('user.create')->middleware('auth');
30  Route::post('uzivatel', 'UsersController@store')->name('user.store')->middleware('auth');
31  Route::get('uzivatel/{user}', 'UsersController@show')->name('user.show')->middleware('auth');
32  Route::get('uzivatel/{user}/edit', 'UsersController@edit')->name('user.edit')->middleware('auth');
33  Route::patch('uzivatel/{user}', 'UsersController@update')->name('user.update')->middleware('auth');
34  Route::delete('uzivatel/{user}', 'UsersController@destroy')->name('user.destroy')->middleware('auth');
35
```

Obrázek 5: RESTful akce pro uživatele ve web.php odkazující na příslušný controller (zdroj vlastní)

Práce se zdroji je v Laravelu velmi jednoduchá. Stačí pro každou akci HTTP metod jako je create, store, apod. vytvořit routu, jak je ukázáno na obrázku 5 a je hotový kompletní RESTful ovladač včetně vyžadovaných parametrů, zapisovaných ve složených závorkách, které identifikují určitý zdroj, což je na obrázku 5 jeden konkrétní identifikátor uživatele, zpravidla jeho id v databázi. [1]

Tyto zdroje nemusí být pouze určitý identifikátor, ale celý objekt Laravelu, tzv Illuminate Request objekt. Tyto speciální objekty obsahují mnoho informací o requestu a jeho součástí jsou i přenášená data, oprávnění a další parametry HTTP requestu, které procházejí jednotlivými vrstvami, kterými je Laravel aplikace obalená, zvanými middleware. Díky middleware lze omezit přístup ke zdrojům a definice vlastního middleware činí implementaci vlastních zabezpečovacích technik poměrně jednoduchou. Laravel nabízí základní middleware, který lze definovat buď v souboru web.php, jak je zobrazeno na obrázku 5. nebo v konstruktoru každého controlleru, kam request putuje. Základním middleware je např. auth, který zabrání přístupu ke zdroji nepřihlášenému uživateli, nebo verified, který zabrání přístupu neověřeného uživatele. Dalším způsobem omezení přístupu je definování tzv. policy, což je organizační struktura pro určení konkrétního přístupu pro každou RESTful akci nad zdrojem. [1]

1.5.5 Controller

Controller neboli řadič je v podstatě třída, která na jednom místě organizuje logiku jedné nebo více rout. Pracuje s modely a veškeré akce, které mohou být vykonány nad určitým zdrojem a jsou vykonávány v jedné z metod, kam je request přesměrován z web.php. Hlavním účelem conrolleru je tedy zachytit podstatu HTTP requestu, provést určité operace, jako je validace, nebo připojení dalších dat k requestu, a poslat ho dále, třeba do databáze v případě uložení, nebo zpět do pohledu v případě chybně zadaných dat. [1]

Vytvořit controller lze pomocí příkazu `php artisan make:controller ExamplesController`. Controllery jsou umístěny ve složce *app/http/Controllers*, ve které je v základní instalaci umístěn pouze jediný, ze kterého všechny vývojářem definované dědí. Každá z metod definovaných v controlleru musí vracet určitou hodnotu, většinou se jedná o zdroje vyžadované pro frontend aplikace, nebo pohledy (view), které jsou uživateli zobrazovány v prohlížeči. Zdroje posílané metodami se připojují pomocí *compact('zdroj')* nebo *with('zdroj').* [1]

1.5.6 View a šablony Blade

Třetí hlavní komponentou MVC architektury je View, neboli pohled, na kterém je uživateli zobrazena prezentace zdrojů zaslaných controllerem v podobě formuláře, obrázku, stavové zprávy, apod. Tyto komponenty jsou umístěny ve složce *resources/views/*. Laravel používá pro pohledy šablony Blade, které vývojáři znatelně usnadní práci se zdroji aplikace a umožní dědit ze základní komponenty *app.blade.php* umístěné v podsložce *layouts/*. Pro tvorbu těchto komponent neexistuje žádný Artisan příkaz, jsou často tvořeny kopírováním existujících šablon jako potomek *app.blade.php* a musí mít koncovku *.blade.php*. [1]

Blade využívá speciálních metod pro manipulaci s obsahem stránky a skripty, mezi něž se řadí *@include* pro zahrnutí jiné šablony, *@section* pro definici určité sekce, *@yield* pro vložení určité sekce, @extends pro dědění z jiné šablony a základní operace s daty, jako jsou cykly (*@foreach/forelse, @empty, @endforeach, @endforelse*) a podmínky (*@if, @else, @endif*). Blade má také speciální funkci *@csrf* pro vložení speciálního tokenu, který musí být součástí každého formuláře. Tento token chrání aplikaci proti cross-site request forgery, neboli podvržení požadavku neznámým zdrojem. Vložení PHP kódu do HTML elementů v šablonách Blade probíhá pomocí dvou složených závorek *{{ php_kód }}* místo *<?php php_kód ?>*. Tento kód je automaticky kontrolován proti skriptovacím útokům. [1]

Struktura Laravel view je tedy skoro totožná s obecným html dokumentem, rozdílem je elegantnější a kratší syntaxe php kódu a rozšířená funkcionalita. Práce se zdroji probíhá pomocí proměnných poslaných do view z controlleru pomocí metod compact nebo with, fungují zde relace a funkce definované v modelech, jelikož proměnné (převedené na již zmíněné Illuminate Request objekty) jsou často data tabulek získaných z modelů, se kterými se díky výše zmíněným speciálním funkcím pohodlně pracuje. [1]

1.5.7 Migrace

Laravel nabízí velmi užitečný koncept migrací tabulek databáze Jedná se o PHP skripty, které jsou schopny měnit strukturu a obsah připojené databáze. Jsou umístěné ve složce *database/migrations/* a vytváří se skrze příkazový řádek pomocí příkazu `php artisan make:migration create_nazev_table -create`, který generuje třídu se dvěma funkcemi, *up()* pro definici nové tabulky pomocí fasády *Schema::create('nazev', definice_blueprint);* a *down()* pro opačnou operaci pomocí *Schema::dropIfExists('nazev');*. Je použito speciálních funkcí pro definici jednotlivých parametrů tabulek, které jsou poté převedeny do SQL dotazů definujících novou tabulku, cizí klíče, a další. Díky tomuto nástroji lze napsat celou strukturu databáze v Laravelu a poté lze pomocí příkazu `php artisan migrate` naplnit databázi novými tabulkami. [1] [2]

```php
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedBigInteger('ciselpod_id')->nullable();
            $table->string('email')->unique();
            $table->integer('active')->default(1);
            $table->string('notification_preference')->default('mail, database');
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->char('role')->default(2);
            $table->rememberToken();
            $table->timestamps();

            $table->foreign('ciselpod_id')->references('id')->on('ciselpod');
        });
    }

    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Obrázek 6: Migrace tabulky uživatelů aplikace (zdroj vlastní)

Asi největší výhodou psaní těchto skriptů je to, že takto napsané skripty fungují pro většinu běžně používaných databázových systémů, mezi něž se řadí MySQL, SQLite PostgreSQL, a lze využít knihoven třetích stran pro další méně používané, například Firebird. Odpadá nutnost přepisování skriptů z dialektu jedné databáze do dialektu druhé. Stačí spustit výše uvedený příkaz s přepínačem *--database=nazev_jineho_pripojeni_k_databazi* nebo vyměnit přístupové údaje k databázi v *.env*. [2]

Při vytvoření migrace je novému souboru přiděleno jméno dle času spuštění příkazu následujícím způsobem: *rok_mesic_den_cislo_create_nazev_table.php*. Časové údaje slouží pro určení pořadí tvorby tabulek, kde tabulka s vazbou na jinou tabulku pomocí cizího klíče musí mít pozdější časový údaj, jinak by došlo k chybě stejně jako při spuštění klasického SQL skriptu. Pokud pořadí nesedí, soubor stačí pouze přejmenovat s jiným časovým razítkem. [2]

1.5.8 Eloquent

Abstraktní vrstva Eloquent je databázový nástroj postavený na standardních SQL dotazech. Podobným způsobem jako migrace, poskytuje rozhraní pro interakci s různými databázovými systémy pomocí jednoho dialektu. Jedná se o ActiveRecord ORM, česky objektově relační zobrazení aktivních záznamů, což znamená, že jedna Eloquent třída může představovat celou tabulku, stejně jako jeden určitý záznam v tabulce. [1]

Samotný nástroj staví na jednoduchosti a, stejně jako celý framework Laravel, na co nejkratším kódu. Obyčejný select všech řádku tabulky z obrázku 6 do proměnné *$users* lze provést pomocí *$users = User::all();,* kde User je název modelu reprezentující tabulku users. Pokud se nejedná o jeden konkrétní řádek tabulky, je obsahem proměnné *$users* tzv. kolekce, kterou lze definovat jako obal pole objektů, kde objekt je jeden řádek tabulky, a nabízí řadu užitečných metod práce s těmito daty. Výběr a vložení jednoho řádku tabulky do proměnné *$user* lze provést pomocí *$user = User::where('id',1)->firstOrFail();.* Pro přístup k datům jiné tabulky pomocí dialektu Eloquent je potřeba mít definované relace v modelu, které byly popsány v kapitole 1.5.3 Model včetně příkladu použití. [1]

1.5.9 Frontend

Laravel jako takový je především backendový framework, ale poskytuje řadu komponent zaměřených na frontend, přidávaných pomocí komponenty jménem Laravel Mix. Jedná se o uživatelské rozhraní postavené na programu Webpack, což je nástroj pro kompilaci různých JavaScriptových modulů a závislostí. V kořenové složce projektu je umístěn soubor webpack.mix.js, který slouží jako konfigurační rozhraní pro Webpack. V základní instalaci Laravelu obsahuje pouze pár řádků, které Webpacku říkají, kde hledat veškeré skripty, což je soubor *resources/js/app.js*, který je srdcem veškerého JavaScript kódu Laravel aplikace. [1]

Pokud vývojář nechce frontend vyvíjet odděleně např. v Angularu, nabízí Laravel jako startovací bod tvorbu frontend části pomocí frameworků Bootstrap, React a / nebo Vue. Novější verze nemají tyto nástroje ve standardní instalaci implementované, a musí se přidat do závislostí projektu pomocí příkazu `composer require laravel/ui`. Po přidání závislostí lze vybrat jeden z výše uvedených, a lze i generovat standardní přihlašovací modul pomocí příkazu `php artisan ui bootstrap –auth`. Pro použití Vue v aplikaci Laravel stačí pouze do app.js přidat řádek *window.Vue = require('vue');* a práce s frameworkem je možná pomocí klíčového slova Vue. Příkladem práce s Vue je vytvoření nové komponenty *MojeVueKomponenta.vue* ve složce *resources/js/components/* a registrace této komponenty v *app.js* přidáním kódu *Vue.component('moje', require('./components/MojeVueKomponenta.vue').default);*. Následuje kompilace nového kódu pomocí příkazu *npm run dev* (více o npm a jak tento příkaz zprovoznit v kapitole 3.5). Pokud byla kompilace úspěšná, lze v jakémkoliv view použít novou komponentu pomocí <*moje*> přímo v kódu aplikace. [1]

2 ALTERNATIVNÍ PHP FRAMEWORKY

Tato kapitola je věnována rešerši alternativních PHP frameworků a jejich základnímu srovnání dle vybraných kategorií, které mohou pomoci s výběrem frameworku z různých hledisek.

2.1 Symfony

Framework Symfony byl zmíněný v předešlých kapitolách, protože řada jiných frameworků včetně Laravelu využívá jeho výborné stability a podpory, která je bezkonkurenční. Vývoj probíhá pomocí komponent zaměřených na specifickou problematiku jako je routing, konfigurace, tvorba formulářů a autentifikace, které činí výslednou aplikaci dále rozšiřitelnou. Nabízí velké množství vlastností a funkcí, které lze samostatně implementovat i do jiných frameworků jako je Laravel a Zend, a vlastní testovací nástroje. Je určen pro pokročilé vývojáře a velké podniky, proto není doporučován pro začátečníky v oblasti webového vývoje. [10][11]

2.2 CakePHP

CakePHP byl jedním z prvních frameworků na trhu stavěných na MVC architektuře. Je jedním z nejlehčích na naučení, především kvůli jeho CRUD implementaci a lehké instalaci, jelikož potřebuje pouze webový server. Jeho přednostní kvalitou je obrovské množství stáhnutelných komponent pro spoustu těžko implementovatelných funkcí a kódovací konvence zaměřené na rychlý vývoj. Tyto konvence je třeba striktně dodržovat, což může být bráno jako nevýhoda. Je dobrou volbou pro komerční aplikace díky kvalitní implementaci bezpečnostních technik a prémiové podpoře pro podniky. [8] [9]

2.3 CodeIgniter

Výkonný a jednoduchý framework CodeIgniter využívá MVC architektury, je určený pro rychlý vývoj webových aplikací. Jedná se o velmi lehký framework, který včetně dokumentace zabírá pouze okolo dvou megabajtů. Nabízí řadu funkcí pro tvorbu robustních a znovu použitelných komponent pro řešení specifických úloh v aplikaci. Díky rychlé instalaci, velké volnosti a dobré dokumentaci je vhodným pro začátečníky. Zaostává za konkurencí v četnosti aktualizací a oficiální podpory, proto není vhodný pro aplikace vyžadující vysokou úroveň zabezpečení. [9] [11]

2.4 Zend

Zend framework užívá modulární architekturu, čímž si získal označení „slepovaný framework“. Využívá objektově orientovaného paradigmatu programování a umožňuje vývojářům použít jen potřebné komponenty v podobě knihoven a ignorovat vše ostatní. Díky otevřenosti kódu lze vyvíjet vlastní samostatné komponenty, použitelné v jiných frameworcích jako knihovny. Nehodí se pro začátečníky, protože je potřeba mít dobré znalosti OOP a návrhových vzorů. Je vhodný pro velké podnikové projekty a je upřednostňovaným frameworkem technologických společností včetně Microsoftu, Adobe, IBM a Google. [9] [10]

2.5 Srovnání frameworků

Tabulka 1: Srovnání frameworků (zdroj vlastní)

Název	Laravel	Symfony	CakePHP	CodeIgniter	Zend
Jednoduchost	3	1	3	3	1
Rychlost práce	3	2	2	2	1
Dokumentace	3	3	2	2	3
Vyspělost a	2	3	2	2	2
Benchmark [4] [5]	2	3	2	2	3
Popularita [6] [7]	3	2	1	2	1
Aktualizace a	3	3	2	1	3
Celkem	19	17	14	14	14

Popsané frameworky byly srovnány dle kategorií, které jsou pro vývojáře důležité při rozhodování, který si vybrat pro svůj projekt v hodnotící škále 1-3 body. Nebyly zahrnovány kategorie typu práce s databází, testování a zabezpečení, protože porovnání těchto je velmi subjektivní a každý ze srovnávaných frameworků má své vlastní řešení. [8] [9] [10] [11]

Nejlépe se umístil Laravel s 19 body z 21 možných, těsně následovaný Symfony se 17 body. V závěsu se umístily CakePHP, CodeIgniter a Zend se 14 body.

3 ZÁKLADNÍ TECHNOLOGIE PRO PRÁCI S FRAMEWORKEM LARAVEL A SOUVISEJÍCÍ

V této kapitole jsou představeny základní technologie pro vývoj webové aplikace ve frameworku Laravel včetně databázového systému pro uložení dat a webového serveru pro spuštění aplikace v internetovém prohlížeči.

3.1 Composer

Prvním krokem pro vytvoření projektu frameworku Laravel je stáhnutí programu Composer. Jedná se o multiplatformní nástroj pro správu závislostí v PHP. Program vývojáři umožňuje vytvářet závislosti mezi knihovnami a balíčky, na kterých bude nový projekt záviset, a sám nainstaluje všechny potřebné požadavky pro jejich instalaci. Composer, na rozdíl od balíčkovacích nástrojů typu Yum nebo Apt, spravuje balíčky knihovny a balíčky ve výchozím nastavení pouze pro náš projekt, konkrétně jsou umístěny v adresáři *vendor/* uvnitř projektu. [14]

Vytvoření nového projektu je s Composerem velmi jednoduché. Ve složce, kde bude projekt umístěn, se pomocí příkazové řádky zadá příkaz `composer create-project --prefer-dist laravel/laravel nazev_projektu`. Composer za nás nainstaluje a nastaví veškeré závislosti a po dokončení instalace můžeme projekt otevřít v preferovaném IDE a začít pracovat. Příkaz nám tedy vytvořil základní projekt frameworku Laravel. Dodatečná instalace balíčků se provádí pomocí příznaku *require.* Například pro instalaci balíčku Passport, který byl použit pro správu REST API webové aplikace, lze zadat `composer require laravel/passport`. Instalované a vyžadované balíčky se zobrazí v souboru *composer.json*, umístěném v kořenové složce projektu, sekce *require* a *require-dev* včetně minimálních vyžadovaných verzí. Ve složce projektu se nachází další soubor *composer.lock*, který slouží pro přesné určení konkrétní verze balíčku pro tento projekt. Když pracuje na projektu více lidí, díky tomuto souboru budou všichni pracovat se stejnými verzemi balíčků, aby nedošlo k nekompatibilitě. Aktualizování verzí balíčku v tomto souboru na nejnovější dostupné verze se provede pomocí příkazu `composer update`, který zároveň tyto stáhne do projektu. [15]

3.2 HTML

HTML je hypertextový značkovací jazyk pro tvorbu World Wide Web stránek. Jeho tvůrcem je Tim Berners-Lee, stejně jako celé služby WWW. Vytvořil ho v roce 1991 za účelem šíření informací, které můžou být snadno zobrazeny pomocí internetového prohlížeče. Ve verzi 1.0 byl vydán v roce 1993, ale oficiálně vyšel až v roce 1995 pod jménem HTML 2.0 a záhy poté verze 3.0. Rok 1997 zaznamenal novou verzi 4.0, která přinesla podporu do stránek vložených skriptů, kaskádových stylů a multimediálních objektů. Dlouhodobým standardem se stala verze 4.01, která vyšla koncem roku 1999 a opravovala některé chyby. Postupně je nahrazována novějším standardem HTML 5, vydaném s oficiálním doporučením W3 v roce 2014. [13] [17]

Struktura HTML dokumentu se dělí na tři základní části, jimiž jsou deklarace verze HTML, head a body. Tyto tagy jsou nepovinné, i bez jejich přítomnosti se struktura dokumentu dodržuje stejně. Část head obsahuje informace o dokumentu, jako jsou nadpis, přidané kaskádové styly a další data, která nejsou považována za obsah. Do body části se vpisuje obsah stránky, tedy text, grafika, audio a další. [18]

Jazyk se skládá ze značek (tzv. tagů) a jejich atributů, které slouží ke strukturálnímu rozložení stránky a píší se do špičatých závorek. Většinou se jedná o párové tagy, příkladem lze uvést tag *<h1>Veškerý text v tomto bloku je nadpis první úrovně</h1>*. Některé ovšem v párech nejsou, příkladem je prázdný element *<img>*, kde se všechny atributy zapisují před druhou špičatou závorku. Tagy lze rozdělit do tří skupin, strukturální pro rozložení dokumentu, popisné, které alterují obsah dokumentu a stylistické. Jsou identifikovány dvěma způsoby, pomocí *id* a *class* atributů. Identifikátor *id* musí být unikátní v celém dokumentu, zatímco *class* může být přiřazen více elementům. Díky těmto způsobům identifikace lze efektivně vytvářet styly jak pro jedinečný element, tak pro celou třídu elementů, které můžou být kombinovány. Framework Bootstrap, použitý pro stylistickou stránku této práce, využívá identifikátoru *class*, do kterého se vpisují klíčová slova, která tímto způsobem upravují vzhled jednotlivých elementů. [18]

3.3 PHP

Skriptovací jazyk PHP je jedním z nejrozšířenějších jazyků pro tvorbu backend části webových aplikací. V roce 1995 byla uvolněna jeho první verze, a to s otevřeným zdrojovým kódem. Druhá verze disponovala nástroji pro práci s formulářovými prvky, a komunikaci s databázovým serverem. Důležitým mezníkem je verze 5, která vylepšila podporu objektově orientovaného paradigmatu programování. Verze 5.3 přinesla jmenné prostory, anonymní funkce, lexikální uzávěry a další pokročilé programovací techniky využívané v moderním vývoji. Nejnovější verze je v současnosti 7.4.6. [13] [26]

Do dnešního dne je použit ve skoro osmdesáti procentech webových stránek. I když jeho popularita částečně ustoupila, díky frameworkům, jako je Laravel nebo Symfony, je minimálně na stejné úrovni jako konkurence. Největší konkurencí jazyka je stále ASP.NET, i když s pouze deseti procenty použití na webových stránkách. Mezi další konkurenční jazyky pro serverovou část se řadí Ruby, Java. Python, nebo JavaScript, které se pohybují pod čtyři procenta. [13] [24]

PHP kód se vpisuje do HTML kódu pomocí tzv. PHP tagů ve formě *<?php echo $var; ?>*. Jelikož je PHP serverový jazyk, jsou veškeré příkazy provedeny ještě předtím, než je stránka stáhnuta klientovi, díky čemuž lze snadněji docílit podpory různých internetových prohlížečů. [27]

V jazyce PHP se proměnné zapisují pomocí symbolu *$* (znak dolaru), název nesmí obsahovat mezery a může obsahovat velká i malá písmena, číslice a podtržítka. První znak za symbolem $ může být jen podtržítko nebo písmeno. Datový typ proměnné jazyk PHP rozpozná dle dat vložených do proměnné a může to být jeden z následujících:

- Integer – celá čísla. Mohou být kladná, záporná, nebo nula.
- Float – desetinná čísla rovněž kladná, záporná, nebo nula.
- String – textové řetězce, což jsou posloupnosti písmen, čísel a speciálních znaků. Textové řetězce jsou zapisovány do uvozovek (ať už jednoduchých, nebo dvojitých).
- Boolean – pravdivostní hodnota; jedna ze dvou hodnot – true nebo false. Údaj nesmí být uzavřen do uvozovek, jinak by se stal textovým řetězcem.
- Array – pole; tj. víceúrovňová datová struktura, která se podobá tabulce.
- Objekt – objektová proměnná. Pro vložení objektu musí být nejdříve deklarována třída, což je struktura, která může obsahovat atributy a metody.

- NULL – datový typ bez hodnoty. Je výchozí hodnotou pro proměnné, do které nejsou vložena žádná data.
- Resource – slouží pro uložení reference (odkazu) na funkce a zdroje mimo PHP. [15]

3.4 CSS

Kaskádové styly vznikly v roce 1996. Jedná se o jazyk pro popis způsobu zobrazení elementů na stránkách. Cílem použití CSS je umožnit návrh a vzhled oddělit od struktury webové stránky a jejího obsahu. Popisuje, jakým způsobem by měly být HTML, XHTML nebo XML elementy formátovány. Implementace CSS se provádí pomocí vytvoření pojmenované skupiny deklarovaných vlastností, které se pomocí selektoru umístí na vybrané elementy. Příkladem takového pravidla může být selektor *a,* kterým jsou nastavována pravidla deklaračního bloku pro HTML element odkaz. Jednotlivé vlastnosti a jejich hodnoty jsou zapsány do složených závorek deklaračního bloku a oddělují se středníkem. Například změna barvy odkazu na černou barvu lze provést přidáním vlastnosti *color* a hodnotou *black*. Existují tři způsoby přidání kaskádových stylů do dokumentu. Externí CSS přidáme pomocí přidání HTML elementu link do hlavičky stránky. Interní CSS se vpisuje do hlavičkového HTML elementu style a třetí variantou je vložení pravidel přímo do vlastnosti style vybraného elementu. Vývojář může využít existujících frameworků jako jsou Bootstrap nebo Tailwind CSS. Aplikování stylů pomocí těchto frameworků se provádí pomocí použití existujících klíčových slov, které lze najít v dokumentaci. [13]

3.5 JavaScript

JavaScript je multiplatformní skriptovací jazyk vytvořený v roce 1995. Syntakticky patří do rodiny jazyků C/C++/Java ale přesto, že je slovo Java součástí názvu, je tomu pouze z marketingových důvodů. Funkčně jde o úplně jiný jazyk. Veškeré animace, pohyblivé části, nebo změna obsahu bez obnovení stránky, jsou možné právě díky těmto technologiím, mezi nimiž je JavaScript nejpopulárnější. Díky standardizovanému rozhraní DOM (Document Object Model) umožňuje přístup k jednotlivým HTML elementům, které lze dynamicky měnit. Jazyk má velmi široké použití i pro validaci formulářových dat, dotazování do některých NoSQL databází, nebo při vývoji her. Díky široké škále možností použití lze celou aplikaci napsat v JavaScriptu. Nevýhodou implementace tohoto jazyka ve webových stránkách je možnost zakázání provádění těchto skriptů uživatelem, které může vést k nesprávnému fungování aplikací. Proto se obvykle spouští až po stažení webové stránky klientem. Kód jazyka, skript, lze zapisovat přímo do HTML dokumentu pomocí tagu *script.* Externě lze skripty přidat pomocí připojení dokumentu s příponou *js* v HTML tagu script. [13]

JavaScript se používá pro dynamičnost aplikace, které lze docílit pomocí zpracování zdrojového kódu na straně klienta v internetovém prohlížeči, na rozdíl od ryze serverového PHP. Jelikož má JavaScript široké použití, lze jej použít i pro vývoj serverové strany aplikace. Pro spuštění JavaScriptu na serveru je třeba mít instalované vývojové prostředí Node.js, dostupné zdarma pro Windows, macOS a Linux. Node.js využívá asynchronních volání, kde, na rozdíl od PHP, není server blokován requestem uživatele, například otevřením souboru, a může přijímat další requesty. Přijatý request vrátí uživateli Promise, tedy slib, že operace ještě nebyla provedena, ale její výsledek je v budoucnosti očekáván. Jakmile je request proveden, server pošle odpověď klientovi, například obsah požadovaného souboru. Takto vznikají tzv. SPA, jednostránkové aplikace (Single Page Application), kde proto není potřeba otevřít novou stránku pro změnu obsahu stránky. Jelikož vše probíhá na jedné stránce, uživatel nestahuje obsah celé stránky znovu a znovu, ale pouze její určité části. Jednostránkové aplikace působí velmi plynulým dojmem, a to díky efektivnější správě paměti, postupnému vykreslování, jak jednotlivé requesty vracejí uživateli slíbené zdroje. [30]

Jako je tomu pro další zde popisované technologie, existují JavaScriptové frameworky, které poskytují obecné řešení funkcionality nezbytné pro vývoj webové aplikace, s možností selektivní změny komponent pomocí vývojářova kódu. Mezi nepopulárnější se dlouhodobě řadí React od Facebooku, Angular od Googlu, a také Vue.js, který je vyvíjen komunitou. Každý z těchto frameworků přistupuje k vývoji trochu jiným způsobem a nelze říci, že je jeden lepší než druhý. Laravel má zabudovanou podporu pro frontend využití Reactu a Vue.js, jelikož Angular je spíše samostatný, plnohodnotný framework s předinstalovanými knihovnami, zatímco React je lehká JavaScript knihovna, kde má vývojář mnohem větší svobodu, které knihovny si vybere pro vývoj. Vue.js je nejmladší z výše zmíněných a využívá to nejlepší z obou. [31]

Každý Node.js projekt začíná s vytvořením souboru package.json, obsahující základní údaje o aplikaci a její závislosti, podobně jako composer.json pro Composer zmíněný v kapitole 3.1. Laravel tento soubor vytvoří automaticky při inicializaci projektu, a je sám jednou ze závislostí. Ať už je používán React, Vue.js či jiný framework pro frontend část aplikace, kód je kompilován do jednoho JavaScriptového souboru *app.js*, v Laravelu umístěném ve složce *public/js/*. Jedná se o obrovský soubor o desítkách tisíc řádků, kompilovaný ze všech uživatelských komponent a závislostí *package.json*. Pro správu závislostí Node.js je používán program NPM (Node Package Manager), instalovaný společně s Node.js. Přidání nové závislosti do projektu se provádí pomocí příkazu `npm install nazev_balicku` do příkazové řádky v kořenové složce projektu. Kompilace je inicializována příkazem `npm run dev`. S výsledným souborem by se nemělo v žádném případě manipulovat, jednak proto, že každá nová kompilace generuje nový soubor, a také proto, že jedna chyba v souboru způsobí nefunkčnost frontendové části pro všechny uživatele. Proto se vývojářův kód vkládá do jiného *app.js* souboru, v Laravelu umístěném ve složce *resources/js/*, kde se také registrují komponenty vytvořené ve vybraném JavaScript frameworku. Pokaždé, když je změněna část kódu, nebo přidána nová komponenta, je třeba znovu spustit kompilaci, jinak nebudou změny viditelné, jelikož nejsou ve zkompilovaném *app.js*. Proto existuje příkaz `npm run watch`, který čeká na změny provedené uživatelem a hned kompiluje nový *app.js* soubor, čímž odpadá nutnost opakovaného zadávání `npm run dev`. [30]

3.6 Java

Programovací jazyk Java vznikl v roce 1995 jako malý jazyk, který se vešel na jednu disketu a obsahoval pouze 211 veřejných tříd. Verze 1.1 z roku 1997 přinesla vnitřní a vnořené třídy a zdvojnásobila počet veřejných tříd. Verze 1.2 přinesla přepracovanou knihovnu a řady dalších knihoven a počet veřejných tříd stoupl na 1524. Změny byly tak zásadní, že se nové verzi jazyka začalo říkat Java 2. Java 2 verze 1.3 pokračovala v evoluci a přinesla další rozšíření knihovny, verze 1.4 přinesla navíc rozšíření syntaxe. Další důležitou verzí ve vývoji byla verze 1.5 z roku 2004, o které současné materiály hovoří jako o verzi 5.0. Přinesla velké množství důležitých rozšíření a počet všech tříd překročil hodnotu 15 000. Označení nových verzí odpustilo od označení tečka nula a nová verze Java SE 6, která s sebou přinesla i nové rozhraní pro tvoření grafických formulářů s názvem Swing, který byl v roce 2008 nahrazen novějším JavaFX. Nejnovější verzí je Java Se 14, vydaná 17. dubna 2020 a stejně jako předchozí verze přináší další drobná vylepšení a opravy chyb předešlých verzí. [22]

Java je multiplatformní programovací jazyk, nezávislý na operačním systému a hardwaru. Mezi specifické rysy Javy patří jednoduchost, a proto je to jeden z první jazyků vyučovaných na univerzitách. Jedná se o objektově orientovaný programovací jazyk. Tato technologie je v současnosti považována za hlavní proud programování. Patří mezi tzv. hybridní jazyky, kdy je program nejdříve přeložen do „mezijazyku“, který je poté interpretován na cílové platformě. Díky těmto vlastnostem je Java dlouhodobě jedním z nejpopulárnějších programovacích jazyků pro tvorbu formulářových aplikací, her a mobilní aplikací jako je i vyvinutá aplikace pro zapsání vlastních odečtů vodoměru. [22]

3.7 Databáze

Před samotnou definicí databáze je potřeba definovat, co jsou to data. Data jsou kolekce informací, které mohou mít různou podobu a formu textu, čísel, mediálních souborů a mnoho dalších. Databáze je organizovaná kolekce strukturovaných dat, jejímž účelem je umožnit data zpřístupnit, spravovat a upravovat. První databáze začaly vznikat v 60. letech 20. století, kdy se začínalo s přesouváním dat z papírových kartoték na datová uložiště. V roce 1970 E. F. Codde publikoval první relační databázi, který pohlíží na strukturu dat formou tabulek. [13]

Na tabulku lze nahlížet jako na běžnou dvourozměrnou tabulku, která má pevně daný význam jednotlivých položek, sloupců. Záznamy jsou uloženy ve formě řádků. Všechny tabulky mají společnou jednu zásadní věc, měly by obsahovat sloupec pro jednoznačnou identifikaci každého záznamu, označovaný jako primární klíč, bez kterého by s nimi nešlo relačně pracovat. Klíče jsou esenciální vlastností tabulek a díky nim vznikají mezi jednotlivými tabulkami relace. Tyto relace jsou vytvářeny pomocí speciálních sloupců tabulky zvaných cizí klíče, které odkazují na hodnotu primárního klíče (nebo jiného unikátního identifikátoru) v jiné tabulce. Relace se rozlišují na čtyři základní kategorie. Relace jedna ku jedné vyjadřuje spojení dvou tabulek, kde záznam jedné tabulky může mít přiřazen pouze jeden záznam tabulky druhé. Příkladem této relace může být průkaz studenta, který patří vždy jednomu a pouze jednomu studentovi. Další relací je jedna ku N a obráceně N ku jedné, kde záznam v jedné tabulce může být přiřazen více záznamům v tabulce druhé. Tuto relaci si lze představit jako knihu, kde kniha může mít více stránek, zatímco stránka patří pouze do jedné knihy. Poslední relace M ku N může být vyjádřena pomocí vztahu studenta a předmětu, kde student může studovat více předmětů a předměty mohou být zapsány více studenty. [29]

Dotazovací jazyk SQL je používán pro vytváření, vkládání, vyhledání, upravování a mazání záznamů uložených v databázi a mnoho dalších operací. Tyto dotazy většinou rozdělují do tří kategorií, a to na DDL, DML, DCL. Některé zdroje uvádějí i čtvrtou kategorii, TCL.

DDL poskytuje příkazy pro databázová schémata, popisy a jak jsou data v databázi umístěna. Pomocí CREATE lze vytvořit databázi nebo její objekty jako jsou tabulky, indexy, procedury a další. Pro upravování těchto ALTER a pro odstraňování DROP. Patří sem i TRUNCATE, který vymaže data tabulek, ale ponechá ve schématu tabulku samotnou, COMMENT pro přidávání komentářů do datového slovníku a RENAME pro přejmenování objektu.

Příkazy kategorie DML slouží k manipulaci s daty. Mezi nejznámější příkazy patří SELECT pro vybírání dat databáze za účelem zobrazení, INSERT pro vkládání dat do tabulek, UPDATE pro změnu existujících dat a DELETE pro odebrání záznamů.

DCL je asociováno výhradně s právy a povoleními databázového systému. Do této kategorie patří příkaz GRANT pro přidělení oprávnění, REVOKE pro odebrání privilegií přidělených prvním zmíněným příkazem a DENY pro zakázání určitých privilegií.

Poslední zmiňovanou kategorií jsou příkazy z rodiny TCL, které slouží pro správu transakcí databáze. Transakce je převod databáze z jednoho konzistentního stavu do druhého a je vykonána pouze pokud nové změny neobsahují žádné chyby. Příkaz COMMIT vytvoří nový konzistentní bod otevřením nové transakce, zatímco ROLLBACK umožní vrácení k předchozímu konzistentnímu bodu. Pokud první příkaz zaznamená chybu na posledním řádku kódu, jsou všechny změny vráceny příkazem druhým. [20] [21]

3.8 MySQL

MySQL je více vláknový řídící systém relační databáze neboli RDBMS. První verze se objevila na trhu v roce 1995 a systém prošel dlouhým vývojem až k dnešní nejnovější verzi 8.0. Díky otevřenému zdrojovému kódu a dvojímu typu licencování, pod GPL i plně komerčním řešením, si získal velkou oblibu. Je dostupný na více než 20 platformách. Systém MySQL využívá model client-server. Jádrem je MySQL server, který zpracovává veškeré instrukce a příkazy posílané MySQL klientem instalovaném na jednom, či více počítačích, kterým poté zobrazí výsledek. Přístup do systému je řízen dle úrovně přístupu a použití přístupového hesla. Využívá standardní formu SQL jako dotazovacího jazyka. [19]

3.9 Php Storm

Jelikož programování v PHP, JavaScriptu, nebo jen čistém HTML nepotřebuje kompilátor, který provádí překlad vývojářova kódu do nižšího programovacího jazyka, nebo strojového kódu, lze využít i obyčejného textového editoru. Ovšem použití speciálních vývojových prostředí (IDE) přináší řadu výhod (a také nevýhod, např. rychlost), které umožňují použít jeden program pro editaci kódu, podporu verzování, nápovědy, formátování a mnoho dalšího. Mnou vybraný PHP Storm české společnosti JetBrains je jedním z nejpopulárnějších nástrojů pro vývoj webových aplikací. Díky vestavěné podpoře pro řadu frameworků, včetně Laravelu, je velmi efektivním nástrojem, který navíc umožňuje manipulaci s databázemi a příkazovou řádkou, tedy skoro vše, co je k modernímu vývoji potřeba. Je dostupný ve 30denní bezplatné zkušební verzi, nebo s licencí na jeden rok pro studenty, a to pro operační systémy Windows, macOS a Linux. [28]

PRAKTICKÁ ČÁST

První kapitola praktické části je věnována popisu technologií použitých při vývoji aplikací. V dalších kapitolách je probrán návrh aplikací, který zahrnuje popis funkčních a nefunkčních požadavků, návrh databázového modelu, popis tabulek, UML diagram aktivit a UML diagram případů užití. Poslední dvě kapitoly popisují implementaci obou aplikací včetně obrázků nejdůležitějších stránek a aktivit.

4 APLIKAČNÍ ŘEŠENÍ

Tato kapitola popisuje návrh vyvíjených aplikace. V první části jsou popsány vybrané technologie použité pro vývoj aplikací a jejich testování. Návrh samotných aplikací vychází z funkčních a nefunkčních požadavků reálné vodárenské firmy, která poptává klientský portál pro zapsání vlastního odečtu a prohlížení základních údajů odběrných míst. Následuje návrh databázového modelu, diagram případů užití, diagram aktivit, a shrnutí návrhu aplikací.

4.1 Technologie použité při vývoji

Tato kapitola představuje mnou vybrané nástroje pro vývoj, včetně vývojového prostředí, webového serveru a programu pro interakci a REST API.

4.1.1 API

API je rozhraní pro programování aplikací, které umožňuje vzájemnou komunikaci mezi dvěma aplikacemi. Je nezbytnou součástí jakékoliv aplikace na mobilu, která se připojuje k internetu. Všechna nestatická data, která jsou zobrazena například v aplikaci pro sledování počasí v mobilu, nová upozornění na sociální síti nebo poslání zprávy v chatu, jsou operace probíhající přes API. Co ale skutečně API poskytuje, je bezpečnostní vrstva. Server a aplikace spolu komunikují v malých dávkách pomocí přesných a určitých requestů a sdílí pouze nezbytné informace. Pomocí internetu aplikace zašle data na server, ten je přijme, zpracuje a pošle odpověď zpět. Aplikace tato data opět zpracuje a zobrazí v čitelném formátu. Tímto requestem může být právě zjištění aktuálního stavu počasí, kdy aplikace zašle request, který server vyhodnotí a vrátí aktuální data z meteostanice. [3]

Existuje několik architektonických stylů webových API, nepopulárnějším a také autorem použitým je REST nebo RESTful API. REST lze použít na téměř všech protokolech, ale typická webová aplikace využívá protokolu HTTP, protože není potřeba instalovat žádné dodatečné knihovny nebo další software. API postavené na protokolu HTTP používá pět základních dotazovacích metod pro manipulaci se zdroji na webovém serveru. Jedná se o GET pro získání zdroje z webového serveru, POST pro předání, PUT a PATCH pro upravení a DELETE pro jeho odstranění. Podrobněji se těmto metodám věnuje kapitola 0. Má pět klíčových vlastností a jednu volitelnou:

- Klient a server by měli být oddělené a schopné provádět změny samostatně.
- Každý klientský request obsahuje veškeré nezbytné informace potřebné pro porozumění requestu serverem.
- Využití cache k ukládání předchozích odpovědí a umožnění rychlejší komunikace díky menšímu množství přenesených dat.
- Univerzální rozhraní, standardizovaná komunikace mezi klientem a serverem.
- Vrstvení, kde ke skutečnému přístupu na server musí request projít jednou nebo více zabezpečovacími vrstvami, jako je proxy a firewall.
- Zasílání kódu, nebo apletu pro jeho použití v aplikaci, které umožní aplikaci modifikovat a vytvoří tzv. chytrou aplikaci. Tato vlastnost je volitelná a využívána nejméně.

Díky těmto vlastnostem lze vytvářet velmi komplexní a zároveň výkonné a flexibilní rozhraní a je na vývojářovi do jaké hloubky jednotlivé vlastnosti využije ve svém API. [3]

4.1.2 Android Studio

Vývojové prostředí Android Studio je společných dílem amerického gigantu Google a české vývojářské společnosti JetBrains. Jedná se o software určený pro vývoj mobilní aplikací pro platformu Android. Je zcela zdarma a dostupné pro platformy Windows, macOS a Linux. Podporuje vývoj v jazycích Java, C++ a Kotlin, který je od května 2019 preferovaným jazykem společnosti Google pro platformu Android. [23]

Aktivita je základní komponentou aplikace, definuje jeden pohled, například přihlašovací formulář. Po přihlášení je zobrazena navazující aktivita. Přepínání mezi jednotlivými aktivitami se provádí pomocí Intentu, ke kterému lze přidat proměnné, objekty nebo celé struktury. Návrh aplikace využívá značkovacího jazyka XML, kde se nastavuje rozložení, vlastnosti a identifikátory jednotlivých komponent umístěných v aktivitě. Existuje celá řada balíčků pro práci s REST API, převaděčů Json nebo XML, grafického vykreslování a mnoho dalších. Sestavení výsledné aplikace je prováděno pomocí nástroje Gradle. Pro testování funkčnosti aplikace lze využít emulátoru, který spustí vybraný virtuální mobilní telefon nebo tablet v počítači, nebo lze připojit vlastní mobilní telefon a instalovat aplikaci přímo. [23]

4.1.3 Postman

Původně pouze rozšíření pro prohlížeče, program Postman, dostupný pro Windows, OS X a Linux, slouží k návrhu a interakci s HTTP API. Získal velkou popularitu počínaje rokem 2012, kdy byl dostupný v Chrome Web Store. Mimo jiné umožňuje psaní automatických testů, monitoring a vytváření API requestů. [16]

Program je velmi přehledný a díky historii všech zaslaných HTTP requestů jsem si ho vybral pro testování svého API. Má jednoduché rozhraní. Vybere se jeden z HTTP requestů, zadá se přístupový bod webové aplikace a vyžadované parametry. Odpověď serveru se zobrazí v dolní polovině aplikace včetně HTTP kódu odpovědi.

4.2 Požadavky

Následující tabulky popisují funkční a nefunkční požadavky webové aplikace. Požadavky reálné firmy byly rozšířeny o administrátorskou část umožňující definici nových dat jako jsou firmy, uživatelé a odběrná místa, které by pro vodárenskou firmu již existovaly ve vlastním informačním systému. Dalším rozšířením požadavků je vývoj API, které umožňuje stahování údajů o vodoměru do mobilní aplikace.

4.2.1 Funkční požadavky

Tabulka 2: Funkční požadavky webové aplikace (zdroj vlastní)

F1	Nahlášení spotřeby	Aplikace musí umožňovat zápis uživatelských odečtů
F2	Zobrazení údajů	Aplikace musí umožňovat zobrazení základních přehledů uživatelských odečtů, odběrných míst a údajů zákazníka
F3	Vyhledávání	Aplikace musí umožňovat vyhledávání v přehledových tabulkách
F4	Prezentace spotřeby	Aplikace musí umožňovat prezentaci spotřeby grafickou formou a interaktivní tabulkou
F5	Export odečtů	Aplikace musí umožňovat export odečtů do jiné databáze
F6	Správa tabulek	Aplikace musí umožňovat CRUD operace pro uživatele, zákazníky, odběrná místa, vodoměry a odečty odběrných míst
F7	Role uživatelů	Aplikace musí mít implementované uživatelské role administrátor a běžný uživatel
F8	Ověření identity	Aplikace musí mít implementovanou funkcionalitu pro ověření identity pomocí emailu
F9	Vytvoření uživatele	Nový uživatel musí být vytvořen administrátorem a přiřazen existujícímu zákazníkovi
F10	Politika hesel	Aplikace musí umožňovat změnu a kontrolu kvality hesla

Tabulka 3: Funkční požadavky mobilní aplikace (zdroj vlastní)

F1	Přihlášení	Aplikace musí umožňovat přihlášení do aplikace stejnými údaji jako webová aplikace
F2	Identifikace vodoměru	Aplikace musí umožňovat identifikaci vodoměru pomocí skenování unikátního identifikačního čísla.
F3	Výběr vodoměru	Aplikace musí umožňovat vybrání konkrétního vodoměru
F4	Změna vodoměru	Aplikace musí umožňovat změnu vybraného vodoměru
F5	Nahlášení spotřeby	Aplikace musí umožňovat nahlášení nového stavu vodoměru
F6	Správa odečtu	Aplikace musí umožňovat správu uživatelských odečtů
F7	Přehled spotřeby	Aplikace musí zobrazit grafickou prezentaci spotřeby

4.2.2 Nefunkční požadavky

Tabulka 4: Nefunkční požadavky webové aplikace (zdroj vlastní)

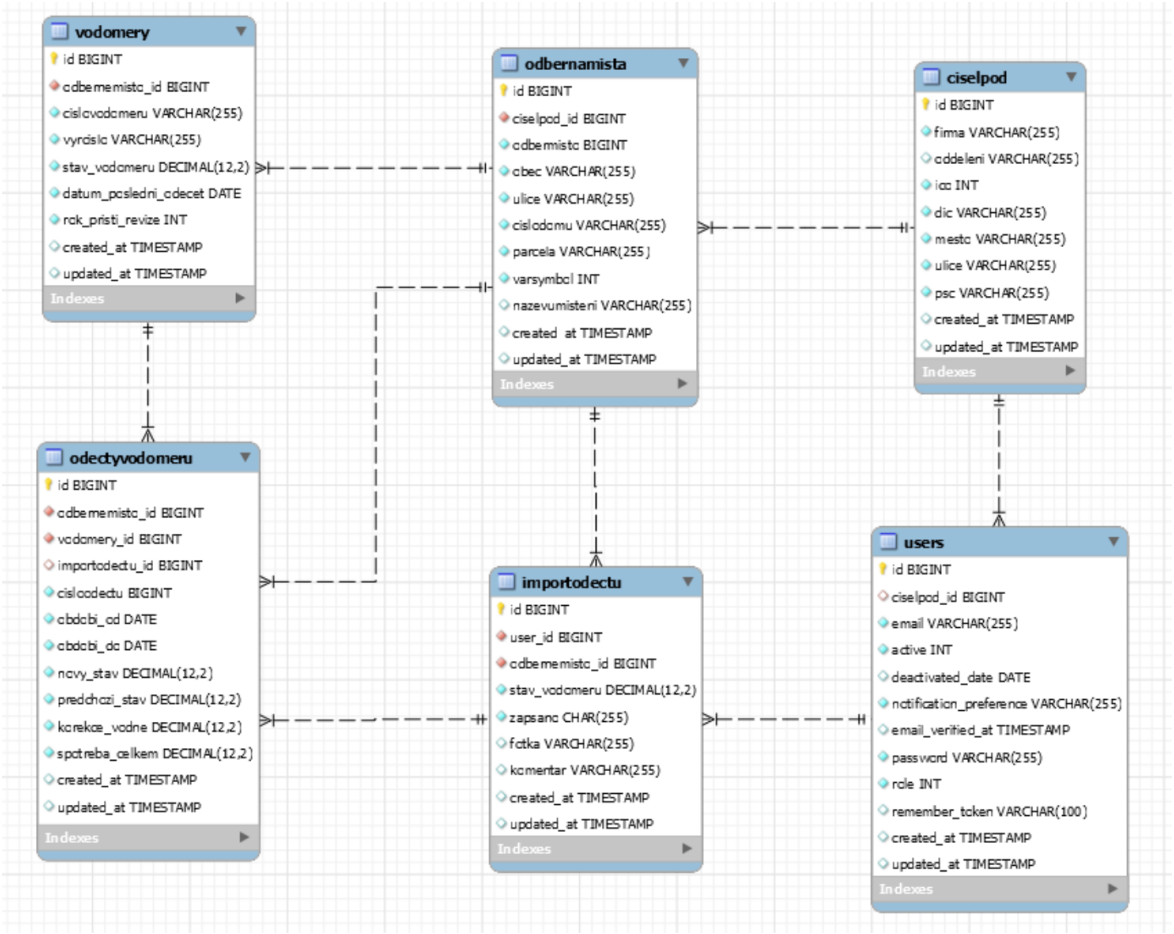
N1	Registrace	Aplikace nesmí umožňovat vlastní registraci uživatelů
N2	Validace	Aplikace nesmí umožnit zadat chybné údaje uživatelských odečtů
N3	Umístění	Aplikaci bude možné umístit na Linux a Windows server
N4	Design	Aplikace musí být responzivní a uživatelsky přívětivá
N5	Bezpečnost	Aplikace nesmí umožnit podvržení požadavku z jiné stránky
N6	Přístupy	Uživatel musí mít přístup pouze ke svým údajům
N7	Editace	Aplikace nesmí umožňovat editaci zapsaných uživatelských odečtů
N8	API	Aplikace musí umožňovat přístup k údajům z mobilní aplikace

Tabulka 3: Nefunkční požadavky mobilní aplikace (zdroj vlastní)

N1	Autentifikace	Aplikace využívá API webové aplikace pro ověření přihlašovacích údajů a stažení údajů vodoměru
N2	Validace	Aplikace využívá API webové aplikace pro validaci zadávaných dat
N3	Platforma	Aplikace musí být spustitelná na platformě Android
N4	Uložiště	Aplikace nemá vlastní databázi ani uložiště

Na základě těchto požadavků je proveden návrh systému. Prvním krokem je analýza podstatných jmen užitých v požadavcích. Tyto údaje jsou klíčem k vytvoření databázového modelu. Analýza sloves užitých v požadavcích je použita pro vytvoření diagramu případů užití a diagramu aktivit. Ty jsou popsány v následujících kapitolách.

4.3 Návrh databáze



Obrázek 7: Databázový model (zdroj vlastní)

Databázový model vyjadřuje relace mezi jednotlivými tabulkami. Pro webovou aplikaci byl vybrán databázový systém MySQL, protože je zdarma a také výchozí pro framework Laravel.

4.3.1 Tabulky

Jádro aplikace tvoří sestava šesti tabulek zobrazených na obrázku 7. Tabulka *users* je automaticky vytvořena Laravelem při inicializaci projektu a stojí na ní celý autentifikační modul. Definuje uživatele aplikace. Má povinné atributy email a heslo, kterými se uživatel přihlašuje, dále obsahuje atributy pro určení, zda je účet aktivní a kdy došlo k deaktivaci, zda byl email verifikován, bez čehož uživatel nemůže postoupit dále za přihlášení a zda je admin či běžný uživatel. Mezi další atributy patří časová razítka (která jsou tabulkám přiřazena automaticky), ciselpod_id pro přiřazení zákazníka, remember_token pro zapamatování hesla při opětovném přihlášení a notifikační preference.

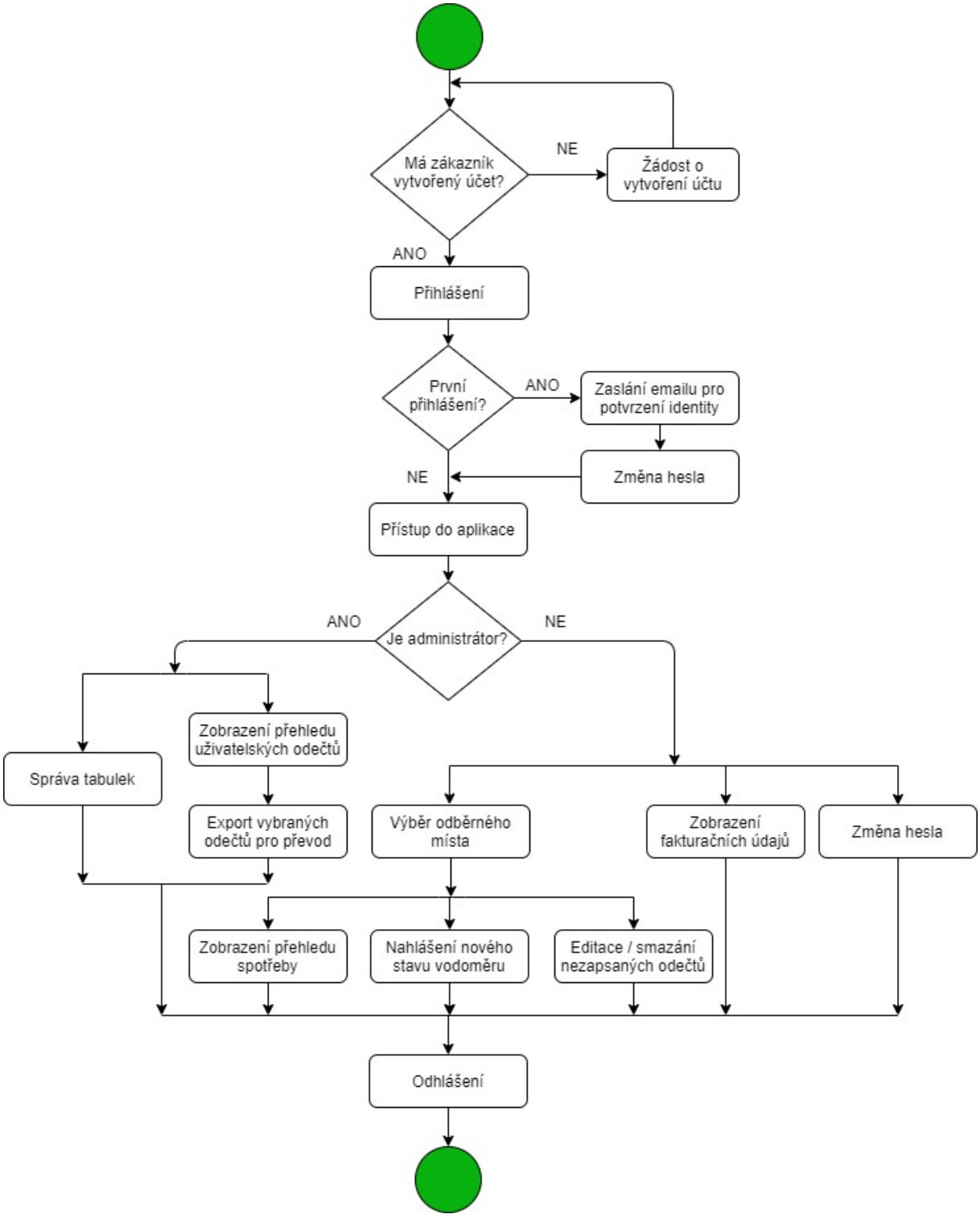
Napojení na podnikový model vodárenské firmy probíhá pomocí relace na *ciselpod*, což je její číselník odběratelů. Pokud chce zákazník mít možnost vstupu do aplikace, musí pro něj být vytvořen záznam v *users* a navázána vazba na *ciselpod*. Obsahuje atributy jako jsou fakturační údaje zákazníka, způsob placení, kontakty, počet dní splatnosti faktur apod.

Další tabulkou je *odbernamista*, která obsahuje příznak na zákazníka. Uchovává údaje o odběrných místech zákazníka, jako jsou adresa umístění a variabilní symbol pro platby. V aplikaci slouží pro základní navigaci, kde uživatel vybere odběrné místo a jsou mu zobrazeny údaje z následujících tabulek. Každé odběrné místo má umístěný vodoměr, ty jsou uchovány v tabulce *vodomery*. Tabulka uchovává datum posledního odečtu a stav zapsaný při odečtu, dále identifikační údaje jako identifikační a výrobní číslo a rok další revize.

Vlastní uživatelské odečty jsou zapisovány do tabulky *importodectu*, která je, kromě změny hesla v tabulce *users*, jedinou uživatelem editovatelnou tabulkou. Má jediný povinný atribut definovaný běžným uživatelem, čímž je stav_vodomeru, ostatní atributy, jako je zapsáno, vazba na odběrné místo a časová razítka, jsou doplňovány automaticky. Volitelnými atributy jsou fotka a komentář, které uživatel může také přidat.

Vazební tabulkou uchovávající historii odečtů odběrného místa a umístěných vodoměrů je *odectyvodomeru*, která také obsahuje příznak na uživatelské odečty, které byly administrátorem převedeny. Uchovává v sobě nový a starý stav, data období, vlastní číslo a případnou korekci vodného a/nebo stočného. Aplikace obsahuje ještě další tabulky, které jsou vytvořeny automaticky při implementaci určitých modulů, bez kterých by nefungovaly. Je jich v mé aplikaci celkem deset. Modul autentifikace potřebuje tabulky pro uchování historie hesel uživatelů a jejich obnovení, modul Passport pro tvorbu REST API jich má hned několik pro uchování přístupů, přístupových tokenů a další operace.

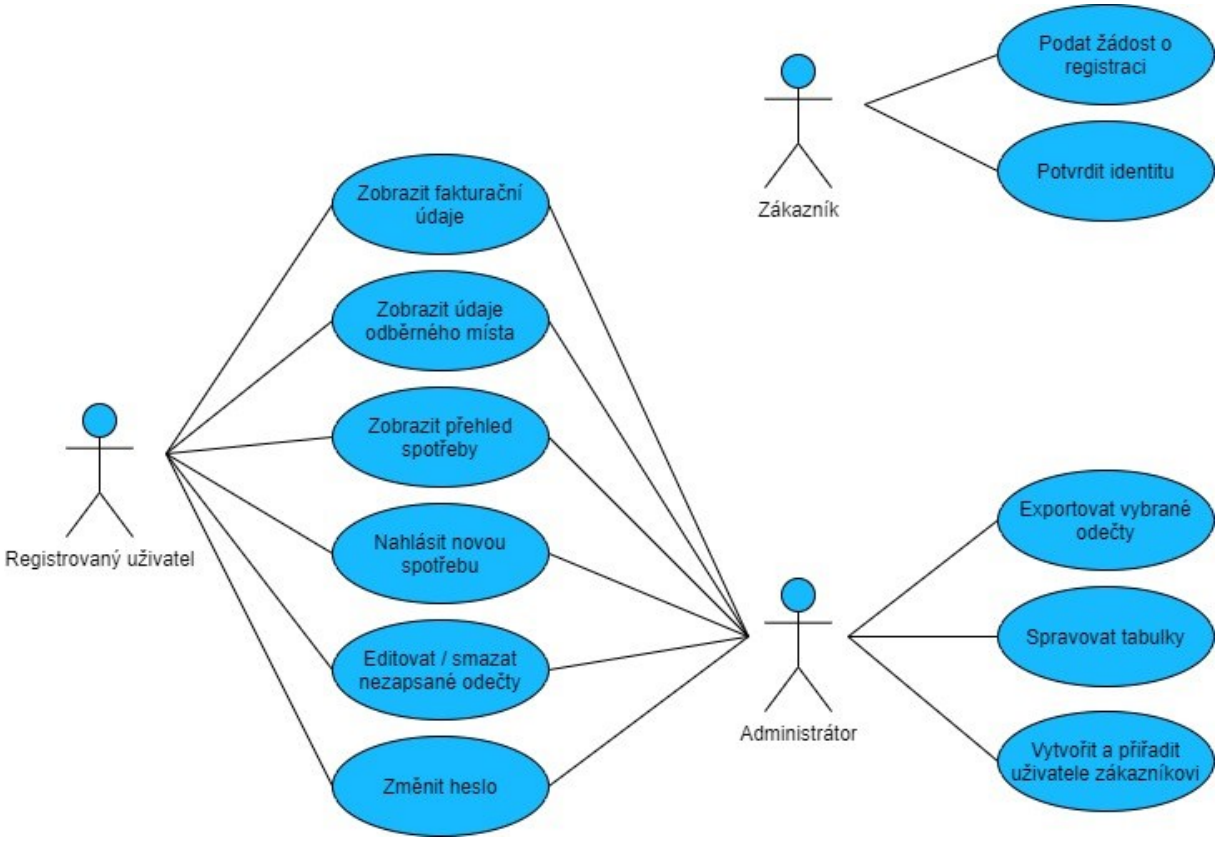
4.4 Diagram aktivit



Obrázek 8: UML Activity diagram (zdroj vlastní)

Diagram aktivit určuje řídící tok aplikace, rozdělený do jednotlivých kroků a procesů. Začíná přihlášením a postupně modeluje možnosti průchodu celou aplikací až k odhlášení.

4.5 Diagram případů užití



Obrázek 9: UML Use case diagram (zdroj vlastní)

Use case diagram, v českém překladu diagram případů užití, vyjadřuje používání aplikace z pohledu uživatelských rolí, jako je přihlášený uživatel, administrátor a zákazník bez uživatelského účtu. Modeluje veškeré možnosti a akce, které uživatel může v aplikaci provádět. Pomáhá vývojáři porozumět požadavkům z pohledu uživatelských rolí v čitelnější podobě. Vlastní diagram modeluje tři aktéry Zákazník, Registrovaný uživatel a Administrátor, kde jednotlivé čáry vyjadřují jejich možné případy užití.

4.6 Shrnutí návrhu aplikací

Aplikační řešení bylo rozděleno do dvou aplikací. Webová aplikace tvoří jádro výsledného projektu a pracuje jako samostatná aplikace, mobilní aplikace využívá dat a funkcionality webové aplikace pro zapsání a zobrazení odečtů. Pro vývoj webové aplikace byl vybrán PHP framework Laravel, protože nabízí kompletní implementovatelné řešení pro popsané požadavky. Umožňuje připojení k více databázím, využívá moderní autentifikace, díky propojení s frontend frameworky lze v jednom projektu vypracovat kompletní řešení a nabízí vlastní řešení pro tvorbu REST API. Pro mobilní aplikaci byl vybrán na základě předchozích zkušeností jazyk Java a Android Studio. Na základě funkčních a nefunkčních požadavků byl vytvořen návrh databáze a byly vytvořeny UML diagramy popisující možné aktivity a uživatelské případy použití aplikace.

4.7 Implementace webové aplikace

V následujících kapitolách bude popsána implementace webové aplikace.

4.7.1 Grafický design

Pro vzhled aplikace byl využit framework Bootstrap, který při správném použití řeší responzivitu za vývojáře. Design aplikací byl navrhnut s důrazem na uživatelskou přívětivost, jednoduchost ovládání a responzivní design. Pracovní plocha je centrována uprostřed stránky tak, aby i na mobilním zařízení byly údaje jasně čitelné a nedošlo k přeplnění obrazovky. Pozadí je klid vyvolávající obrázek moře přecházejícího v oblohu a odstíny modré barvy pro ovládací tlačítka a hlavní panel. Jednotlivé části jsou vytvořeny jako karty a zarovnány pod sebou. Tabulky obsahující zobrazovaná data jsou centrovány a umožňují horizontální rolování daty.

4.7.2 Uživatelské role

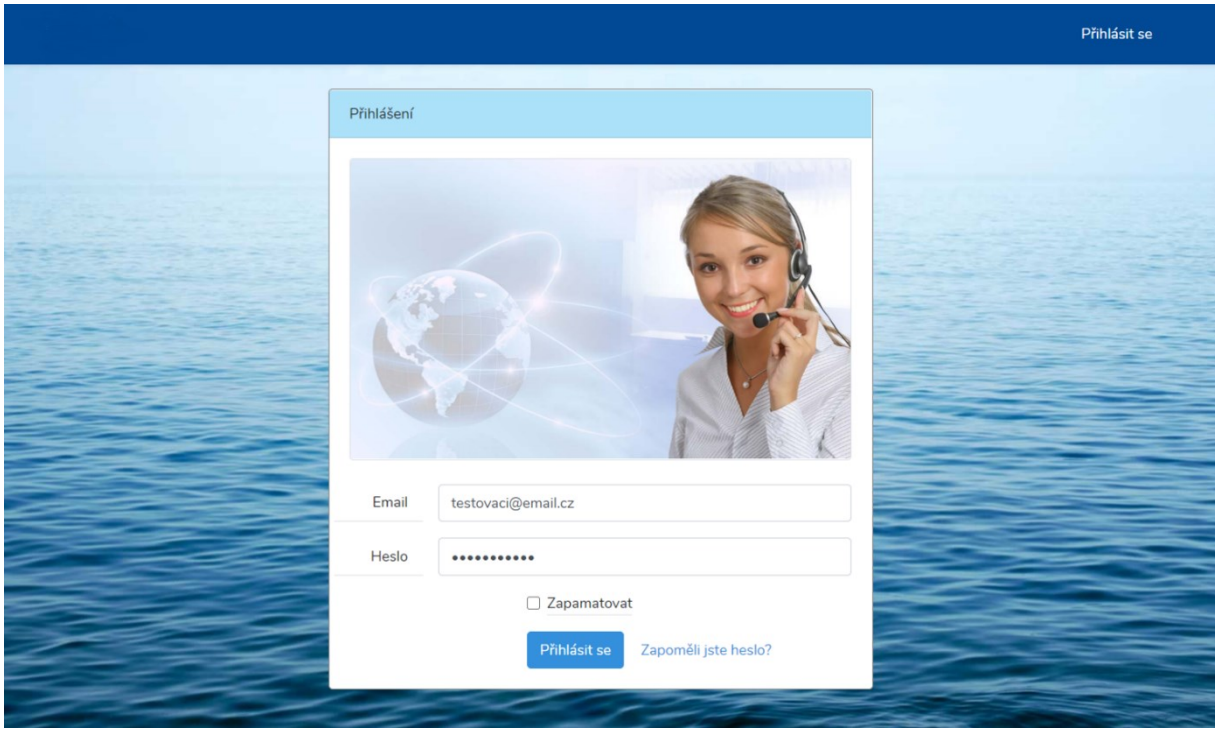
Aplikace je rozdělena na dvě části a funkcionalita odpovídá uživatelským rolím. Administrátorská část umožňuje správu základních tabulek databáze, což znamená tvorbu, editaci a odstranění odběrných míst, vodoměrů, odečtů, zákazníků, uživatelů a jejich vlastních odečtů. Pro tyto operace je přístupná pod navigačním panelem karta Rychlé akce. V reálném provozu by tyto operace byly omezené pouze na tvorbu a správu nových uživatelů pro zájemce o aplikaci z řad zákazníků a export uživatelských odečtů do daného informačního systému.

Běžnému uživateli je přístupná druhá část aplikace, která je především o základních přehledech spotřeby a fakturačních údajů. Umožňuje také nahlášení nové spotřeby svých odběrných míst a správu těchto uživatelsky definovaných odečtů. Řízení přístupu ke zdrojům a omezení pouze na své údaje je nejdůležitější funkcionalitou této části aplikace. Nikdy se nesmí stát, aby uživatel mohl zobrazit údaje jiného uživatele nebo dokonce přistoupit k administrátorské části. Toho je v aplikaci dosaženo ověřením atributu role tabulky users, ověřením relací uživatelského modelu nebo definováním policy v případě přístupu ke zdrojům jiného uživatele.

4.7.3 Přihlašovací modul

Přístup do aplikace je řízen přihlašovacím modulem, na který je nepřihlášený uživatel přesměrován při zadání adresy webové aplikace. Tento modul a jeho základní funkcionalita je generována pomocí příkazu `php artisan ui vue –auth`. Composer stáhne potřebnou funkcionalitu a zbývá pouze úprava chování., např. předělání přihlašování z přezdívky na email a zamezení registrace nových uživatelů, která je dělána striktně na vyžádání zákazníka. Tomu je po vytvoření uživatelského účtu zaslán email s vygenerovaným heslem.

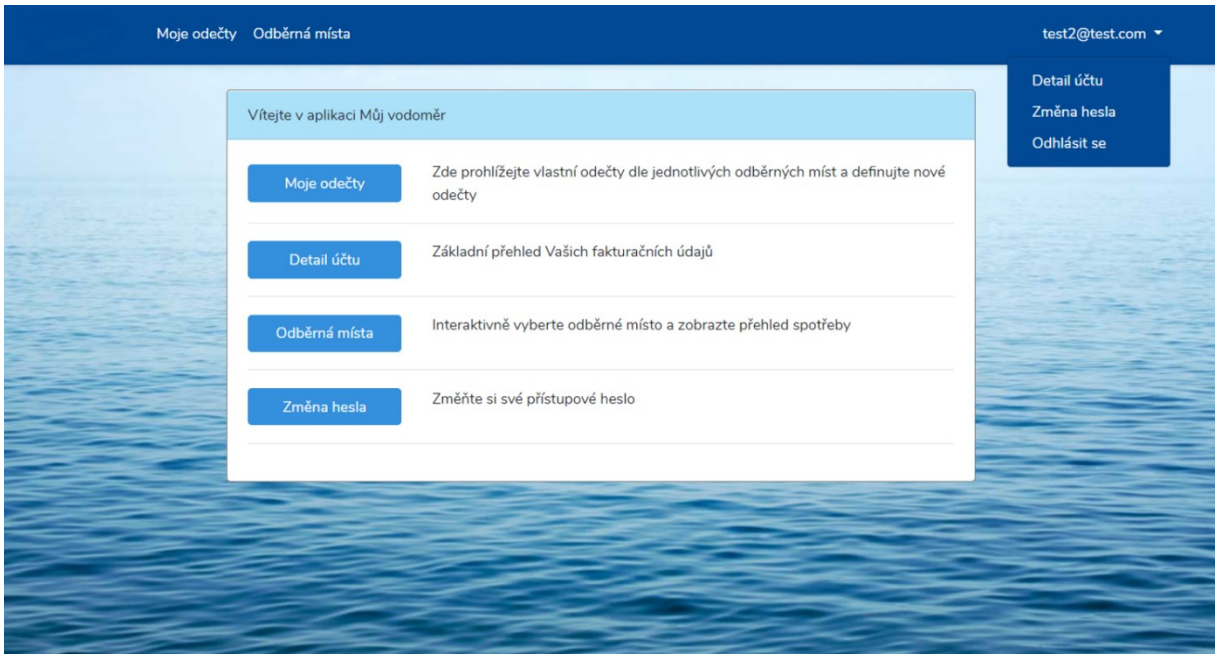
V přihlašovacím formuláři je třeba vyplnit email a heslo. V případě prvního přihlášení je třeba ověřit identitu, na zadaný email je poslán nový odkaz, přes který se uživatel dostane k nastavení nového hesla. Po této verifikaci je uživateli umožněn plný přístup do aplikace.



Obrázek 10: Přihlašovací obrazovka webové aplikace (zdroj vlastní)

4.7.4 Navigace v aplikaci

Po přihlášení je uživateli zobrazena uvítací obrazovka. V horní části se nachází navigační panel umožňující přístup k funkcionalitě aplikace. Na uvítací obrazovce jsou představeny jednotlivé moduly. V pravé horní části je umístěno menu pro správu účtu a odhlášení.

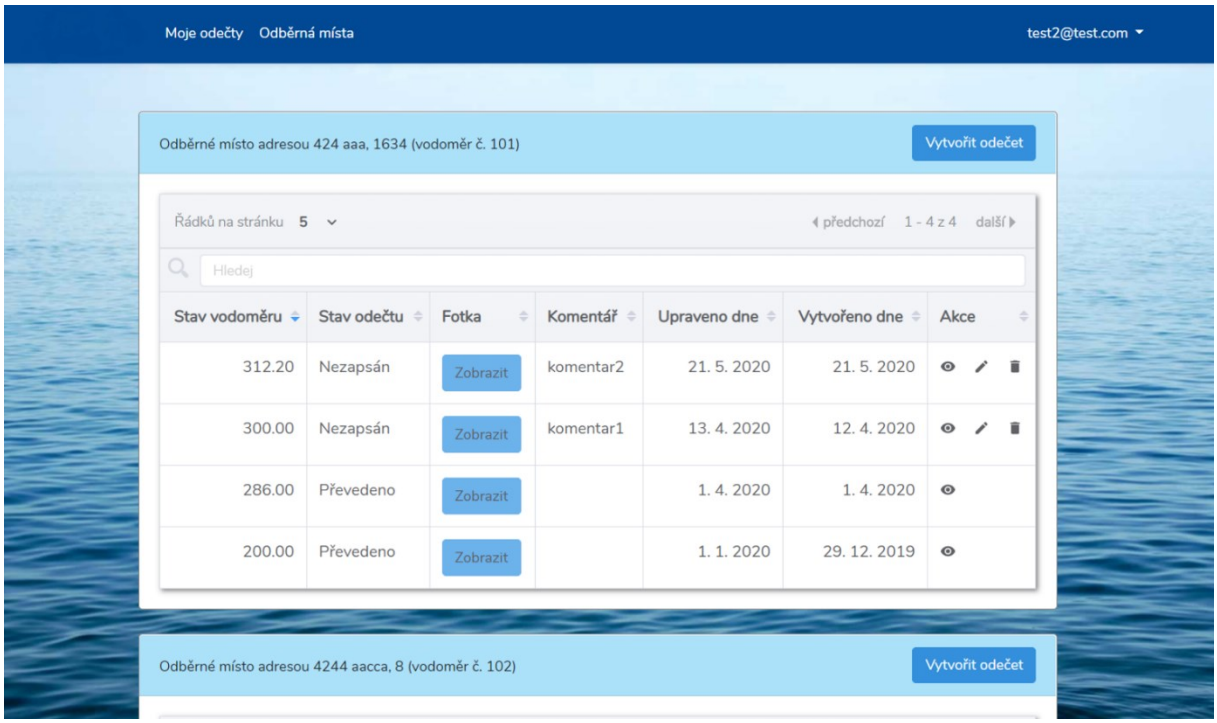


Obrázek 11: Uvítací obrazovka (zdroj vlastní)

4.7.5 Moje odečty

Modul Moje odečty slouží pro rychlý přehled uživatelských odečtů v podobě tabulek. Každé odběrné místo zákazníka má vlastní interaktivní tabulku, která umožňuje řazení sloupců a globální vyhledávání. V pravé části jsou dostupné akce zobrazení detailu odečtu a v případě nezapsaného odečtu (nepřevedeného na skutečný odečet) také úpravu či smazání. Tabulka také obsahuje odkazy na fotky, pokud byly při pořízení uživatelského odečtu nahrány.

Nad tabulkou každého odběrného místa se nachází tlačítko Vytvořit odečet, které uživateli zobrazí formulář s předvyplněnými údaji, jako je číslo vodoměru, poslední stav a datum posledního odečtu. Uživateli zbývá pouze zadat nový stav. K odečtu lze připojit fotku jako důkaz či přidat komentář. Zadaná údaje jsou validovány v příslušném controlleru a pokud je nový stav menší než starý stav nebo je fotka větší než jeden megabajt, je uživatel upozorněn na tyto chyby. Pokud zadané údaje prošly validací, je uživateli zobrazen detail nového uživatelského odečtu a v příslušné tabulce je nový odečet viditelný.



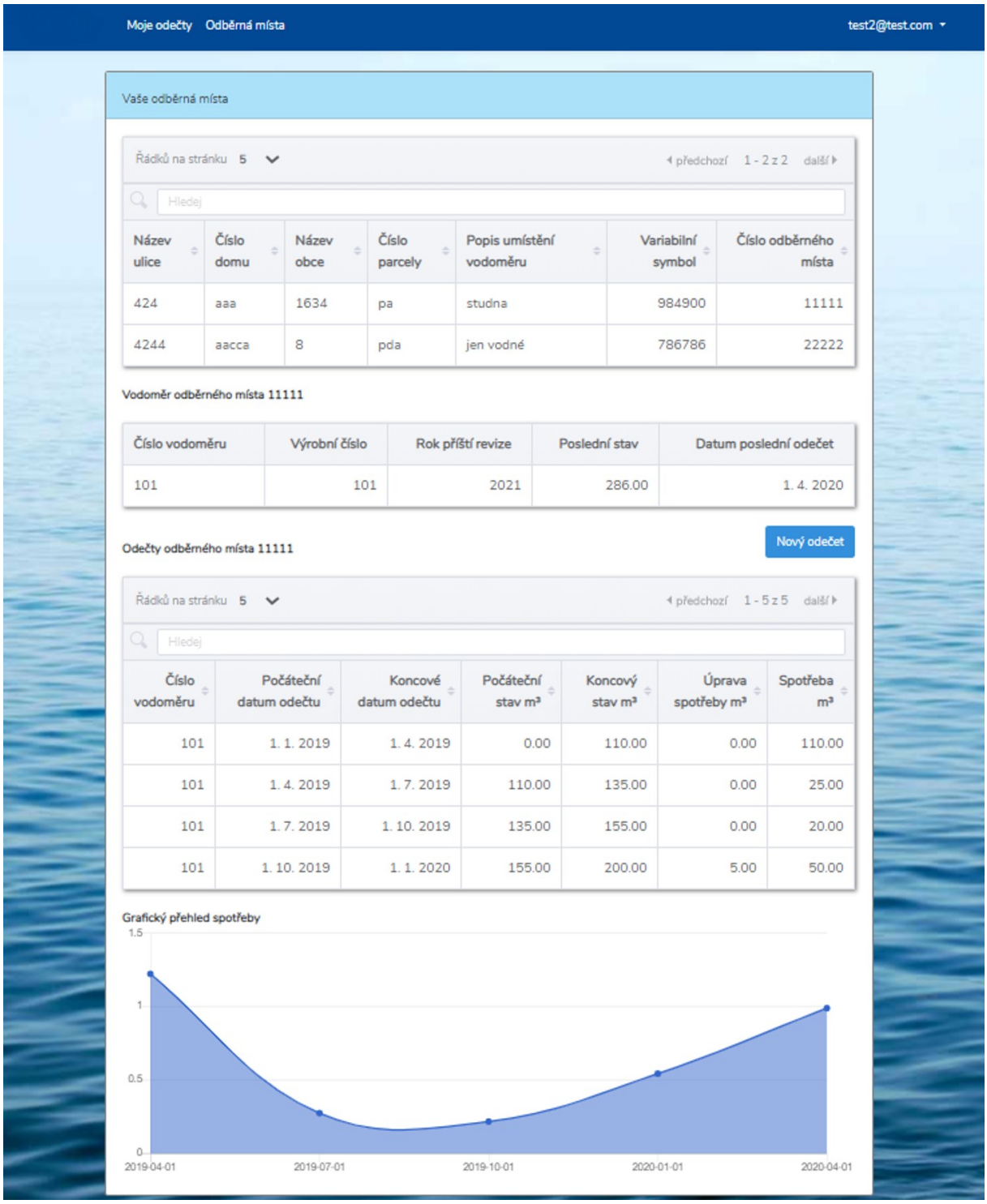
Obrázek 12: Modul Moje odečty (zdroj vlastní)

4.7.6 Přehled spotřeby

Další modul v navigačním panelu je nazvaný Odběrná místa. Je zde umístěn základní přehled odběrného místa, umístěného vodoměru a odečtů včetně grafického přehledu. Po zobrazení této stránky je uživateli zobrazena pouze tabulka jeho odběrných míst. Kliknutím na vybrané odběrné místo je zobrazen zbytek údajů v dalších tabulkách.

Přehled spotřeby vybraného odběrného místa je rozdělen do dvou částí. Interaktivní tabulka je zobrazena hned pod umístěným vodoměrem a zobrazuje kompletní historii odečtů odběrného místa včetně čísla vodoměru a jeho spotřeby. Tabulka tím pádem slouží také jako přehled historie pohybů vodoměrů na daném odběrném místě. Pomocí globálního vyhledávače a řazení sloupců lze v přehledu snadno vyhledávat.

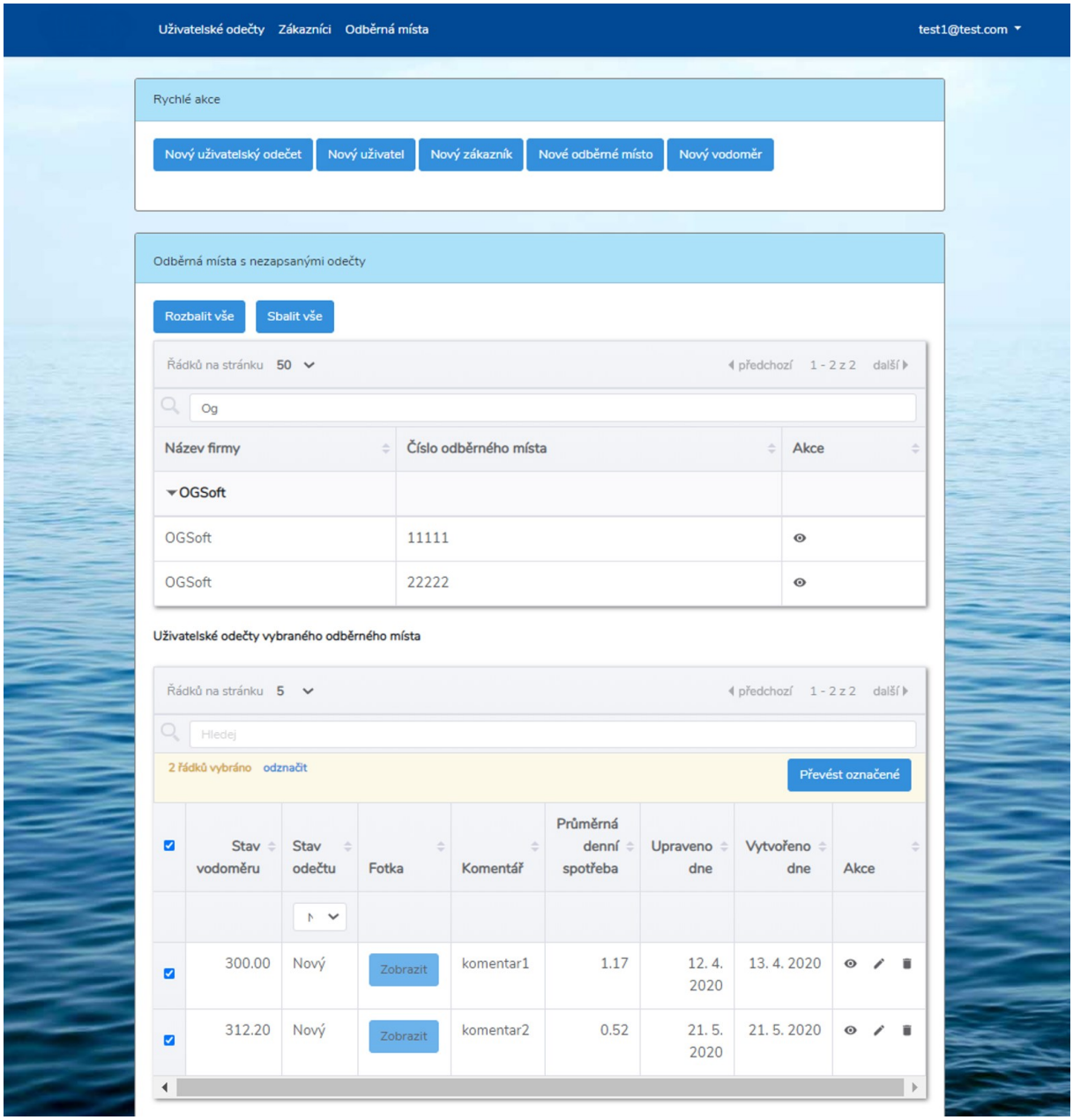
Ve spodní části tohoto modulu je umístěn plošný graf zobrazující průměrnou denní spotřebu. Na X ose jsou umístěna časová razítka jednotlivých odečtů, zatímco osa Y vyjadřuje změnu v průměrné denní spotřebě mezi těmito odečty.



Obrázek 13: Modul Odběrná místa (zdroj vlastní)

4.7.7 Administrátorská část

Uživatel s příznakem administrátor má dostupný modul Uživatelské odečty. V tomto modulu má přístup ke všem uživatelům aplikace, při rozkliknutí detailu lze vybrat odběrné místo, které zobrazí další Uživatelské odečty daného odběrného místa. Tabulka má oproti uživatelské variantě navíc možnost převést vybrané odečty do informačního systému vodárenské firmy.



Obrázek 14: Modul Uživatelské odečty (zdroj vlastní)

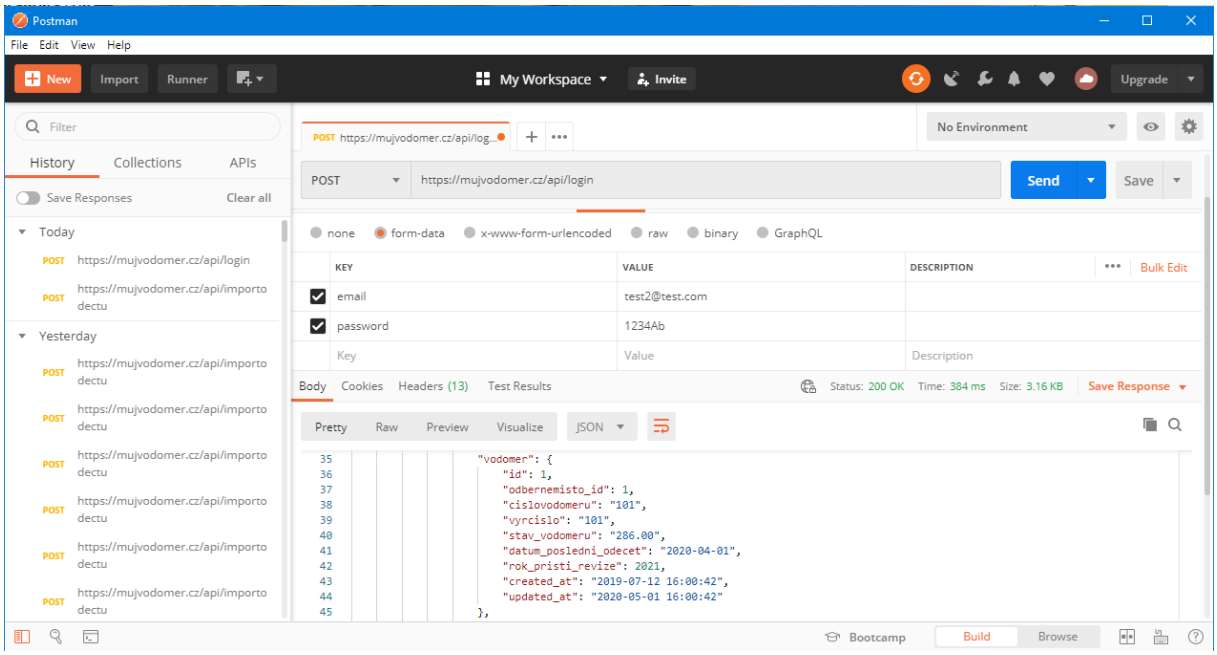
Uživatelský odečet má jeden z příznaků Nový, Exportovaný, Zapsaný a Vyfakturovaný. Nové odečty mají povoleny akce editace, smazání a výběr pomocí checkboxu pro export pomocí tlačítka Převést označené. Odečty s jiným příznakem tyto akce povoleny nemají kvůli integritě dat. Odečty jsou převáděny do databáze informačního systému vodárenské firmy, kde zaměstnanec posuzuje, které odečty zapsat jako skutečné odečty vodoměru.

4.7.8 Tvorba REST API

Komunikace mezi aplikacemi zpravidla probíhá přes API umístěné na serveru, ze kterého klientská aplikace požaduje data. Použitím oficiálního balíčku Laravel Passport lze implementovat vlastní OAuth2 autentifikační server. Instalace probíhá pomocí příkazu `composer require laravel/passport`, který v aplikaci vše potřebné nastaví. Poté je třeba migrovat nové tabulky do databáze a následně pomocí příkazu `php artisan passport:install` vytvořit potřebné šifrovací klíče pro generování zabezpečených přístupových tokenů. Ty mají v případě použití Passportu formu *Bearer access_token*. Tento token se spolu s přihlašovacími údaji přidává do hlavičky requestu z klientské aplikace.

Získání přístupového tokenu bylo implementováno pomocí přihlašovacího controlleru, který ověří uživatelem zadaný email a heslo, a v případě shody je mu zaslána odpověď s tímto tokenem. Ten je třeba si uchovat po celou dobu spojení a přidat do hlavičky requestu s názvem *Authorization* při další komunikaci se serverem.

Pro komunikaci mezi webovou a mobilní aplikací bylo vyvinuto API pro autentizaci pomocí přihlašovacích údajů do webové aplikace a přenos čísel vodoměrů, jejich identifikačních údajů a spotřeby z databáze, a dále REST API pro správu uživatelských odečtů tak, aby z aplikace bylo možné tyto odečty vytvořit, upravit, smazat a stáhnout.



Obrázek 15: Postman - test API (zdroj vlastní)

4.8 Implementace mobilní aplikace

V následujících kapitolách je popsána implementace mobilní aplikace.

4.8.1 Grafický design

Design mobilní aplikace je postaven na relativní pozici jednotlivých komponent, jako jsou tlačítka, textové pole a popisky, k okrajům displeje mobilu. Je tím zaručeno správné zobrazení na většině displejů. Pro pozadí byl vybrán obrázek s odstíny modré až fialové barvy a pro popisky žlutá barva, která je dobře čitelná a na pozadí vynikne. Tlačítka jsou roztáhlá a umístěná pod sebou, aby bylo možné aplikaci ovládat jednou rukou. V každé aktivitě aplikace je zobrazeno číslo aktuálně vybraného vodoměru, aby uživatel vždy věděl, s jakým vodoměrem pracuje.

4.8.2 Připojení k REST API webové aplikace

Mobilní aplikaci lze s webovými API propojit několika způsoby. Jedním z nejpoužívanějších je použití externí knihovny, která poskytuje všechny potřebné nástroje. Mnou vybraná je Retrofit knihovna, která umožňuje vytvoření rozhraní pro HTTP API v jazyce Java a tvorbu requestů. Standartní formát přenášených dat je JSON, který je přímo určený pro výměnu dat. Používá notaci složených závorek pro jasné určení objektů, jejich parametrů a přiřazených hodnot. Každý přenesený objekt má v Javě vytvořenou vlastní třídu, do které jsou tyto objekty uloženy pro plné využití objektově orientovaného paradigmatu programování.

4.8.3 Autentifikace uživatele

První aktivita zobrazená uživateli po spuštění aplikace je přihlášení (obrázek č. 16). Zadáním přihlašovacích údajů a stisknutím přihlašovacího tlačítka je proveden požadavek na server, kde dojde k ověření přihlašovacích údajů s tabulkou users. Pokud Oba údaje souhlasí, je v odpovědi serveru zaslán přístupový token, který je potřeba mít uložený po celou dobu relace. Uživateli je zobrazena krátká notifikace o odpovědi serveru, která může být úspěšné přihlášení či špatné přihlašovací údaje. Pokud není telefon připojen k internetu, nebo je server nedostupný, je uživateli zobrazen http kód chyby a není možné aplikaci používat.

Obrázek 16: Přihlášení do mobilní aplikace (zdroj vlastní)

Obrázek 17: Výběr čísla vodoměru (zdroj vlastní)

4.8.4 Jednoznačná identifikace vodoměru

Po úspěšném ověření zadaných údajů a obdržení přístupového tokenu je uživateli zobrazena další aktivita (obrázek č.17). Zde má uživatel možnost vybrat konkrétní vodoměr ze seznamu a použitím pomocí tlačítka Potvrdit je uživatel přesměrován na hlavní obrazovku aplikace, odkud se může navigovat v aplikaci.

Druhá možnost identifikace vodoměru je pomocí tlačítka Sken čárového kódu vodoměru. Pokud aplikace nemá oprávnění k použití kamery, je potřeba tato oprávnění aplikaci dát. Pro identifikaci je použita zadní kamera. Uživatel musí navigovat kameru na čárový kód umístěný na vodoměru tak, aby byl celý zobrazen na displeji a počkat na ověření. Pokud bylo číslo vodoměru rozeznáno, je uživatel přesměrován na hlavní obrazovku aplikace a v případě, že se nepodařil čárový kód rozeznat, je mu nabídnuto zkusit skenování znovu nebo vybrat vodoměr z nabídky.

4.8.5 Navigace v aplikaci

Hlavní menu zobrazuje tlačítka pro navigaci v aplikaci. Jsou to Nový odečet, Zobrazit uživatelské odečty, Přehled spotřeby a Změnit vodoměr. Po celou dobu je uživateli zobrazeno číslo vybraného vodoměru v horní části aplikace, aby uživatel vždy věděl, s jakým vodoměrem aktuálně pracuje.

Stisknutím tlačítka Změnit vodoměr je uživateli zobrazena předchozí aktivita a je mu opět umožněno skenování či výběr vodoměru ze seznamu.

Z hlavního menu se dá navigovat na přehled uživatelských odečtů pomocí tlačítka Zobrazit uživatelské odečty. Uživateli je zobrazen seznam všech nezapsaných uživatelských odečtů, které vytvořil v mobilní či webové aplikaci. Tyto jsou po výběru editovatelné a smazatelné.

Obrázek 18: Hlavní menu (zdroj vlastní)

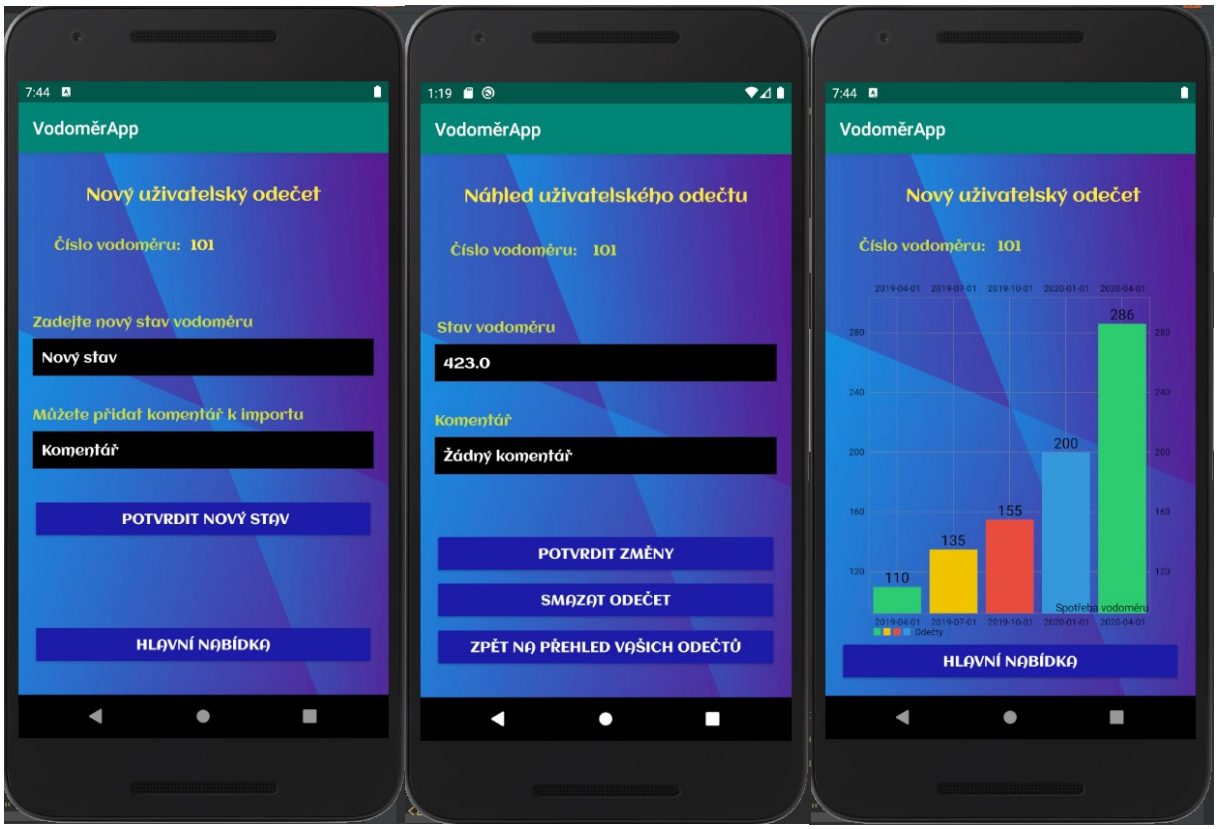
Obrázek 19: Přehled vytvořených odečtů (zdroj vlastní)

4.8.6 Prohlížení a nahlášení spotřeby

Kliknutím na tlačítko Přehled spotřeby je zobrazena aktivita se sloupcovým grafem, která může být při velkém počtu odečtů nepřehledná. Tažením dvou prstů na displeji od sebe je možné přiblížení pro snadnou čitelnost údajů spotřeby. Na ose X lze vidět datum konkrétního odečtu a sloupec grafu zobrazuje nárůst spotřeby. Jednotlivé sloupce jsou barevně rozlišeny pro lepší čitelnost. Pod grafem spotřeby se nachází tlačítko pro přechod na hlavní menu aplikace.

Nahlášení nového odečtu je možné pomocí formuláře přístupného pomocí tlačítka Nový odečet. Uživatel zde vyplní nový stav vodoměru a volitelně také komentář k odečtu. Je zamezeno zadat speciální znaky, aby nemohlo dojít k zaslání potenciálně nebezpečného kódu na server.

Zadaný odečet, spolu s dalšími nezapsanými odečty lze zobrazit navigací z hlavního menu tlačítkem Zobrazit uživatelské odečty, kde je zobrazen seřazený přehled těchto odečtů. Kliknutím na jakýkoliv odečet je zobrazena aktivita, kde je uživateli umožněno vybraný odečet editovat. Časová razítka vytvoření a editace jsou automaticky přidána nebo aktualizována na serveru.



Obrázek 20: Nový uživatelský odečet (zdroj vlastní)

Obrázek 21: Editace uživatelského odečtu (zdroj vlastní)

Obrázek 22: Přehled spotřeby (zdroj vlastní)

ZÁVĚR

Práce vznikla na základě motivace vytvořit aplikace, které lze využít v reálném provozu a nasbírat co nejvíce zkušeností s technologiemi pro tvorbu webových a mobilních aplikací. Byly implementovány reálné požadavky získané od poptávající firmy, která by touto aplikací mohla poskytnout svým zákazníkům možnost nahlášení vlastních odečtů, přehled spotřeby a prohlížení jejich údajů.

Pro splnění těchto požadavků bylo nutné nastudovat nejmodernější technologie vývoje webových aplikací a implementovat poptávanou problematiku. Vybrané technologie byly popsané v teoretické části s důrazem na PHP framework Laravel, který byl použit pro serverovou část aplikace a na které je celá aplikace postavena. Dále byly v teoretické části srovnány alternativní technologie, kterými mohla být výsledná aplikace zpracována a popsány základní technologie pro vývoj s vybraným frameworkem.

Praktická část zahrnuje popis vybraných programů pro vývoj a také technologie použité při vývoji mobilní aplikace, která slouží pro identifikaci vodoměru pomocí čárového kódu, zadání odečtu a základní přehled spotřeby. Cílem vývoje klientské mobilní aplikace byla implementace požadavků pro platformu Android a rozšíření obzoru v problematice programování aplikačního rozhraní pro klientskou autentizaci.

Součástí praktické části je návrh výstupních aplikací včetně funkčních a nefunkčních požadavků, na jejichž základě byl proveden návrh databáze a tvorba diagramu aktivit a případů užití. Praktický výstup práce byl od první analýzy konzultován s odborníky na tuto problematiku. V posledních kapitolách byly popsány výsledné aplikace se všemi možnostmi, které nabízí. Při vývoji byl kladen důraz na bezpečnost, přehlednost, responzivní design a rozšiřitelnost aplikací o další funkčnosti. Výsledné aplikace jsou funkční a splňují zadané požadavky, převážně reálné zadání vodárenské společnosti a jsou připravené pro nasazení do testovacího provozu. Před nasazením do produkce je třeba aplikace vyladit a rozšířit, převážně webovou část, o rozhraní pro použití více vodárenskými firmami na jednom serveru.

Při vývoji aplikací jsem se naučil využít framework Laravel pro rychlý a efektivní vývoj, který byl pro mě nový. Bylo nutné porozumět architektuře MVC a bezpečnosti práce s osobními údaji. Také jsem se naučil vytvářet responzivní design pomocí frameworku Bootstrap a umístění webové aplikace na sdílený hosting. Při vývoji mobilní aplikace jsem se naučil zásady komunikace pomocí internetového protokolu HTTP a práci s přenášenými daty. S výslednými aplikacemi jsem spokojen a vývoji se chci v budoucnosti aktivně věnovat.

POUŽITÁ LITERATURA

[1] STAUFFER, Matt. *Laravel: Up & Running: A Framework for Building Modern PHP Apps*. 2 vyd. Sebastopol: O'Reilly Media, 2019. ISBN: 978-1-492-04121-4.

[2] REES, Dayle. *Laravel: Code Smart*. Leanpub, 2016. 458 s.

[3] SCHMIDT, Douglas & STAL, Michael & ROHNERT, H. & BUSCHMANN, F. Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects. Volume 2 Edition. New York City: Wiley, 2000. ISBN 978-0471606956.

[4] Comparator. In: *PHPBenchmarks.com* [online]. [cit. 2020-06-18]. Dostupné z: http://www.phpbenchmarks.com/en/comparator?phpVersions=php-7.2%7Ephp-7.3%7Ephp-7.4&groups=code-igniter-3.x&components=zend-framework-3.0%7Ecake-php-3.7%7Elaravel-5.8%7Esymfony-5.0

[5] Web Framework Benchmark. In: *TechEmpower.com* [online]. [cit. 2020-06-18]. Dostupné z: https://www.techempower.com/benchmarks/#section=test&runid=ca07ce5b-053c-4d75-b204-53eb5a6c5aeb&hw=ph&test=query&l=zg24n3-f&p=0-0-0-1kw-0&a=2&f=0-0-2tc0-55fd4-ba4g-2hwirk-0-27wr28-4zthd-4fti4g-0

[6] Stack Overflow Trends. In: *stackoverflow.com* [online]. [cit. 2020-06-18]. Dostupné z: https://insights.stackoverflow.com/trends?tags=laravel%2Csymfony%2Czend-framework%2Ccodeigniter%2Ccakephp

[7] Google Trends. In: *trends.google.com* [online]. [cit. 2020-06-18]. Dostupné z: https://trends.google.com/trends/explore?date=all&q=laravel,Symfony,%2Fm%2F02qgdkj,CakePHP,Zend

[8] REIGNS, Stephanie. *11 Best PHP Frameworks for Modern Web Developers in 2020* [online]. [cit. 2020-06-20]. Dostupné z: https://coderseye.com/best-php-frameworks-for-web-developers/

[9] MORRIS, Will. *8 Best PHP Frameworks for Web Developers* [online] [cit. 2020-06-20]. Dostupné z: https://www.hostinger.com/tutorials/best-php-framework

[10] BEREZHNOI, Roman. *Best PHP Frameworks for Web Developement* [online]. [cit. 2020-06-28]. Dostupné z: https://f5-studio.com/articles/best-php-frameworks-for-web-development/

[11] GOUD, Swetha. *PHP Frameworks: Pros & Cons of PHP Framework* [online]. SkillRary, [cit. 2020-06-28]. Dostupné z: https://www.skillrary.com/blogs/read/php-frameworks

[12] STOWE, Mike. *Undisturbed REST: A Guide to Designing the Perfect API*, San Francisco: MuleSoft, 2015. ISBN 978-1329116566

[13] ČEGAN, Lukáš. *Vývoj webových aplikací v PHP a NetBeans*. Pardubice: Univerzita Pardubice, 2015. ISBN 978-80-7395-858-9.

[14] Intro. In: *GetComposer.org* [online]. [cit. 2020-06-28]. Dostupné z: https://getcomposer.org/doc/00-intro.md

[15] HOPKINS, Callum. *PHP okamžitě.* Brno: Computer Press, 2014. ISBN 978-80-251-4196-0.

[16] About Postman. In: *Postman.com* [online]. [cit. 2020-06-28]. Dostupné z: https://www.postman.com/about-postman/

[17] HTML tutoriál. In: *W3cschools.in* [online]. [cit. 2020-06-28]. Dostupné z: https://www.w3schools.in/html-tutorial/

[18] The global structure of an HTML document. In: *W3.org*. [online]. [cit. 2020-06-28]. Dostupné z: https://www.w3.org/TR/html401/struct/global.html

[19] ROUSE, Margaret. *What is MySQL?* [online]. [cit. 2020-06-28]. Dostupné z: https://searchoracle.techtarget.com/definition/MySQL

[20] VAISHNAVI, M R. *What is a Database? Know the Definition, Types & Components* [online]. Edureka, [cit. 2020-06-28]. https://www.edureka.co/blog/what-is-a-database/

[21] SQL Server commands - DML, DDL, DCL, TCL. In*: TechNet.Microsoft.com* [online]. [cit. 2020-06-28]. Dostupné z: https://social.technet.microsoft.com/wiki/contents/articles/34477.sql-server-commands-dml-ddl-dcl-tcl.aspx

[22] PECINOVSKÝ, *Rudolf. Java 14: kompletní příručka jazyka*. Praha: Grada Publishing, 2020. Knihovna programátora (Grada). ISBN 978-80-271-1369-9.

[23] NOVOSAD, Jan. *Programování 2D her v jazyce Java.* Praha, 2016 [cit. 2020-06-28]. Bakalářská práce. Vysoká škola ekonomická. Fakulta in a statistiky. Dostupné z: https://theses.cz/id/8ext94/.

[24] SEMECKÝ, Vojtěch. *Android Studio – nové vývojové prostředí* [online]. 2013 [cit. 2020-06-28]. Dostupné z: https://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/

[25] Usage Statistics and Market Share of Server-side Programming Languages for Websites, June 2020. In: *W3Techs.com* [online]. [cit. 2020-06-28]. Dostupné z: https://w3techs.com/technologies/overview/programming_language

[26] PHP: Releases. In: *Php.net* [online]. [cit. 2020-06-28]. Dostupné z: https://www.php.net/releases/index.php

[27] SUEHRING, Steve & VALADE, Janet. *The Structure of a PHP Scritp* [online]. [cit. 2020-06-28]. Dostupné z: https://www.dummies.com/programming/php/the-structure-of-a-php-script/

[28] *PhpStorm: The Lightning-Smart IDE for PHP Programming by JetBrains*. In: *JetBrains.com* [online]. [cit. 2020-06-28]. Dostupné z: https://www.jetbrains.com/phpstorm/

[29] DRKUSIC, Emil. *Learn SQL: Types of relations* [online]. 2020 [cit. 2020-06-28]. Dostupné z: https://www.sqlshack.com/learn-sql-types-of-relations/

[30] NEMETH, Gergely. *Node Hero* [online]. [cit. 2020-06-28]. Dostupné z: https://blog.risingstack.com/node-hero-tutorial-getting-started-with-node-js/

[31] SIMRAN, Kaur Arora. *10 Best JavaScript Frameworks to Use in 2020* [online]. [cit. 2020-06-28]. Dostupné z: https://hackr.io/blog/best-javascript-frameworks

PŘÍLOHY

Příloha A - CD

Obsah přiloženého CD:

• Plné zdrojové kódy mobilní a webové aplikace.

• Databázový model, obrázek UML Use case diagramu, obrázek UML Activity diagramu.

• Tento dokument.