TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

APLIKACE PRO ZÁKAZNÍKY DISTRIBUCE ELEKTRICKÉ ENERGIE

Bakalářská práce

Studijní program:	B2612 – Elektrotechnika a informatika
Studijní obor:	1802R022 – Informatika a logistika

Autor práce:	**Jakub Pandadis**
Vedoucí práce:	Ing. Tomáš Bedrník
Konzultant:	Ing. Jan Kraus, Ph.D.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jakub Pandadis**
Osobní číslo:	**M14000094**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Informatika a logistika**
Název tématu:	**Aplikace pro zákazníky distribuce elektrické energie**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Na základě závěrů ročníkové práce vytvořte portál pro zákazníky distribuce elektrické energie jako webovou aplikaci pro standardní prohlížeč a mobilní platformy.
2. Pro potřeby návrhu aplikace se seznamte s dostupným databázovým back-endem, pro tvorbu vlastní aplikace použijte vhodný java-scriptový framework, komunikující se serverovým Node.js či jiným a jeho prostřednictvím s databázovým serverem.
3. Vytvořenou aplikaci řádně otestujte.
4. Prezentujte dosažené výsledky a uveďte možnosti dalšího rozvoje aplikace.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **KARPOV, Valeri. Professional angularjs. Indianapolis, IN: John Wiley and Sons, 2014. ISBN 1118832078.**

[2] **SUMMERFIELD, Mark. Python 3: výukový kurz. Vyd. 1. Brno: Computer Press, 2010. ISBN 978-80-251-2-37-7.**

[3] **WOLFGANG, Kirsten. Caché. Vyd. 1. Brno: CP Books, 2005. ISBN 80-251-0491-5.**

[4] **ZAKAS, Nicholas C. JavaScript pro webové vývojáře. Vyd. 1. Brno: Computer Press, 2009. Programujeme profesionálně. ISBN 978-80-251-2509-0.**

[5] **LECKY-THOMPSON, Ed a Steven D. NOWICKI. PHP 6: programujeme profesionálně. Vyd. 1. Brno: Computer Press, 2010. Programujeme profesionálně. ISBN 978-80-251-3127-5.**

Vedoucí bakalářské práce: **Ing. Tomáš Bedrník**
Ústav mechatroniky a technické informatiky

Konzultant bakalářské práce: **Ing. Jan Kraus, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2017**

Termín odevzdání bakalářské práce: **14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2018

Podpis: Pandula Jakub

Poděkování

Rád bych tímto poděkoval všem, kteří se podíleli na zdárném dokončení této bakalářské práce. Zvláště chci poděkovat panu Ing. Tomáši Bedrníkovi za vedení práce a poskytnutí konzultací, panu Ing. Janu Krausovi, Ph.D. za vedení bakalářského projektu, který byl základem pro vznik této práce, a především bych chtěl poděkovat panu Janu Svobodovi, z firmy AlbisTech s.r.o, za poskytnutí rad při řešení problémů ve vývoji aplikační části bakalářské práce.

Abstrakt

Tento dokument popisuje tvorbu webové aplikace, která bude sloužit jako portál pro zákazníky distribuce elektrické energie a bude uživateli poskytovat přehledné informace o odběru elektrické energie v časových úsecích, informace o nastavených limitech, připnutých smluv a dalších dokumentech za dané období.

Bakalářská práce navazuje na bakalářský projekt, který se zabýval návrhem zákaznického portálu a vychází z jeho závěrů. Jelikož se jedná o vývoj webové aplikace, zabývá se první část dokumentu vhodným výběrem javascriptového frameworku, komunikující se softwarovým systémem Node.js, a výběrem vhodné databáze. Po okomentování zvoleného výběru se práce přesouvá k samotnému aplikačnímu vývoji, kde se po krocích, které řeší danou konkrétní problematiku, postupuje až ke konečnému zhotovení kompletní aplikace.

Celá práce je završena řádným otestováním aplikace v provozu a uvedením dalších možností rozvoje.

Klíčová slova:

zákaznický portál, bakalářská práce, webová aplikace, elektrická energie, javascriptový framework

Abstract

This document describes the creation of a web application that will serve customers as a portal for electricity distribution and provide clear information about electricity consumption in time slots, information on set limits, contracts and other documents in a given period.

The bachelor thesis concludes the bachelor project, which treated about the design of a customer portal, and the bachelor thesis is based on those conclusions. For the reason that that is a web application development, the first part of the document is concerned with selecting a javascript framework, communicating with the Node.js software system, and selecting a suitable database. After commencing the selected selection, the work treats about the actual application development, in which particular problems are solved step by step until the finalisation of the complete application.

Completion of all the work is dependent on successful testing of the application in operation, otherwise other development options are proposed.

Keywords:

customer portal, bachelor thesis, web application, electrical energy, javascript framework

Obsah

Seznam obrázků

Seznam obrázků přílohy A

Seznam zdrojových kódů

Seznam zkratek

DOM	Document Object Model, objektově orientovaná reprezentace modelu dokumentu
URI	Uniform Resource Identifier, textový řetězec s definovanou strukturou sloužící ke specifikaci zdroje informací
REST	Representational State Transfer, architektura rozhraní pro distribuované prostředí
HTML	HyperText Markup Language, značkovací jazyk pro tvorbu webových stránek
JSON	JavaScript Object Notation, nezávislý datový formát na počítačové platformě
SVG	Scalable Vector Graphics, značkovací jazyk a formát souboru popisující dvojrozměrnou vektorovou grafiku
API	Application Programming Interface, rozhraní pro programování aplikace
HTTP	HyperText Transfer Protocol, internetový protokol určený pro výměnu dokumentů ve formátu HTML
CLI	Command Line Interface, uživatelské rozhraní ovládající se přes příkazový řádek
URL	Uniform Resource Locator, textový řetězec s definovanou strukturou sloužící ke specifikaci zdroje informací na internetu
SFTP	SSH File Transfer Protocol, protokol zajišťující bezpečný přenos souborů pomocí počítačové sítě

Úvod

Cílem této bakalářské práce je vytvořit webovou aplikaci, která bude zákaznickým portálem pro odběratele elektrické energie u jednotlivých distributorů. Portál by měl uživateli poskytnout přehledné informace o uzavřené smlouvě s distributorem o dodávce elektrické energie a informace o přiděleném odběrném místě. Nezbytnou součástí je vyobrazení spotřebované energie daného odběrného místa, jak ve zvoleném termínu, tak i v téměř reálném čase. Tyto naměřené údaje by měly býti reprezentovány pomocí tabulek a grafů, kvůli přehledné čitelnosti. Uživatel je poté schopen zjistit svůj odběrový profil a dopočítat se tak finančních nákladů za určitý časový úsek, či upravit svůj denní odběr, pro lepší využitelnost. Nesmí zde chybět ani nastavení limitních hodnot odběru elektrické energie, aby se předešlo následným sankcím, které bývají mnohdy finančně velmi náročné. Další možností, kterou by měl portál disponovat, je podání elektronické žádosti o změnu parametrů smlouvy nebo změnu uživatelských údajů, jako je například doručovací adresa při změně bydliště. V neposlední řadě, by měl mít uživatel možnost zobrazení či stažení smluv, faktur, či jiných dokumentů, přímo z portálu.

Pro dosažení zadaného cíle je potřebné zvolení vhodného softwarového prostředí a systému, který dokáže přehledně a jednoduše zobrazovat informace jak na klasickém stolním počítači, notebooku nebo mobilním telefonu. Z tohoto důvodu se práce, ve své první části, zabývá rešerší vhodných javascriptových frameworků a databází, z kterých je poté vybrána jedna kombinace (zde kombinace frameworku Angular s databází Intersystems Caché), která je použita pro tvorbu aplikace. Následující část práce je věnována samotnému vývoji aplikace. V jednotlivých krocích je vždy zaměřeno na jednu konkrétní problematiku, ke které je přiděleno použité řešení.

V závěru se práce zabývá řádným otestováním aplikace a nastíněním dalšího možného rozvoje.

1 Rešerše – výběr frameworku a databáze

Pro rychlejší vývoj složitějších moderních webových aplikací, se v dnešní době vyplatí používat javascriptový framework. Framework je soubor knihoven a zdrojového kódu, který pomocí speciálních tagů můžeme používat k snadnějšímu vývoji aplikace. Vývojář tak není nucen neustále pracně psát zdrojový kód, ale například pro vypsání všech dat z pole použije speciální tag, který ve frameworku odkazuje na tuto funkci. To dělá konečný zdrojový kód aplikace kratší a přehlednější. Další velkou výhodou je kompatibilita vytvořeného kódu napříč všemi prohlížeči. Není tedy potřeba upravovat kód pro správnou funkčnost v každém prohlížeči, jelikož framework to udělá za nás. Většina dnešních frameworků obsahuje i grafické efekty a animace, čímž můžeme vdechnout trochu života do webové aplikace bez většího a zdlouhavého programování. [1]

Využívání frameworku má i svoje stinné stránky, které ale postupem vyvíjením a zdokonalováním technologie mizí nebo mají minimální negativní účinek. Zde můžeme například zmínit pomalejší chod aplikace. Využití frameworku je vždy o něco pomalejší než použití nativního volání, jelikož máme v cestě o jednu vrstvu navíc. Každopádně v dnešní době je, díky optimalizacím frameworků, tento rozdíl skoro nepoznatelný. S rychlostí aplikace souvisí i načítání frameworku, které trvá o trochu déle, jelikož se musí framework stáhnout do prohlížeče, aby bylo možné ho poté využívat. Tento neduh je patrný především na starším hardwaru či pomalých linkách. Další nevýhodou je to, že s velkou pravděpodobností neexistuje žádný framework, který umí všechno, co vývojář zrovna potřebuje, tudíž i když nám ve velké části usnadní práci, stále se vyskytnout případy, kdy je potřeba si některé funkce doprogramovat svépomocí nebo se poohlédnout po pluginech, které dokáží využitelnost frameworku rozšířit. V neposlední řadě je potřeba zmínit, že framework je nové rozhraní, které se, pro využití jeho funkcionality, musíme naučit. [1]

1.1 Nejpoužívanější javascriptové frameworky

V posledních letech se javascriptové frameworky těší velké popularitě mezi komunitou vývojářů. Dokazuje to výzkum, provedený v letech 2016 a 2017 webem Stack Overflow, který je nejpopulárnější mezi vývojáři, kteří zde sdílejí svoje zkušenosti, problémy, řešení a další poznatky ohledně vývoje v jakémkoliv populárnějším programovacím jazyce. Podle výzkumu bylo vybráno několik javascriptových frameworků, které si představíme na následujících rádcích. [2][3]

Angular – především známý jako AngularJS (název jeho první verze), je framework vyvíjený od roku 2009 společností Google, který pro psaní zdrojového kódu aplikace využívá kombinaci jazyků JavaScript a TypeScript (open-source programovací jazyk, který je nadstavbou jazyka JavaScript, rozšiřující ho o statické typování a atributy, známé z objektově orientovaného programování). V dnešní době je Angular nejpoužívanějším frameworkem pro tvorbu jednostránkových webových aplikací a je volně dostupný na webu angular.io pod MIT licencí. Angular funguje na principu vkládání funkcí do html kódů, a tím tak vytvoří dynamický náhled – interaktivní uživatelské prostředí. Pomocí jednotlivých rozšiřujících elementů a atributů, framework vytvoří dynamický html dokument, kde při kompilaci a vykreslení html na uživatelském rozhraní, se v DOM struktuře připojí všechny atributy poskytované direktivami. Jedná se o využití návrhového vzoru Model-View-Controller, kde je většina logiky obsažena v modelu či controlleru, který je poté vložen do zobrazovací šablony, která již je schopna využívat definované proměnné a vypisovat je, či s nimi jakkoliv manipulovat. Hlavní předností Angularu je dvoucestný binding neboli dvoucestná synchronizace dat, kde template (například formulář se vstupy), view (webová stránka) a model (javascriptové objekty) jsou spolu synchronizovány. Pokud dojde k aktualizaci modelu, aktualizuje se i view a tak to funguje i na opačnou stranu. To nám usnadňuje práci při psaní kódu pro manipulaci se strukturou objektového modelu dokumentu. [4][5]

React – je dalším hojně používaným frameworkem a silnou konkurencí pro Angular. React je vyvíjen společností Facebook Inc., ta ho používá pro aplikace Facebook a Instagram. Poprvé byl propuštěn v roce 2013 jako open source pod BSD licencí. Ze

začátku se mezi vývojáři stal terčem kritiky, kvůli „míchání“ html části a části programové, avšak toto nepochopení základního konceptu postupně opadlo a dnes je nejrychleji rostoucím frameworkem, který má četné zastoupení výukových programů a komponent na internetu. V roce 2016 Facebook Inc. uvolnil další verzi Reactu, React Native, který je pro mobilní zařízení využívající systém Android nebo iOS. Tento framework je vysoce výkonný ve vykreslování složitých uživatelských rozhraní, využívá virtuální strukturu objektového modelu dokumentu (DOM), který může být umístěn na straně klienta nebo na serverové straně. V průběhu dochází k porovnávání virtuálního DOM v prohlížeči se skutečným a zjišťují se rozdíly. Po nalezení rozdílu dochází k aktualizaci pouze změněného uzlu DOM, místo vykreslování celé struktury. Mezi jeho další výhody patří znovu využitelnost frameworku, a to ve formě komponent, kde pouze deklarativně definujeme strukturu html skládáním javascriptových funkcí, které následně mohou být používány v dalších aplikacích nebo mohou být sdíleny pro veřejné použití na internetu. [4][6]

Ember – další javascriptový framework využívající návrhový vzor Model-View-Controller. V roce 2011 byl vydán jako open source pod MIT licencí, jeho zakladatelem je Yehuda Katz a je dále vyvíjen Endure Core týmem, kam dnes spadá více lidí. Tento framework vyniká, stejně jako Angular, ve dvoucestným bindingu, a především v tvorbě jednostránkových webových aplikací, kde je kladen důraz na prvky, ve kterých webová aplikace vyniká. Především se jedná o sémantické URI, kde informace na webu, jsou strukturovány a uloženy podle standardizovaných pravidel, což usnadňuje jejich vyhledávání a zpracování. Mezi další přednosti tohoto frameworku patří REST architektura (rozhraní navržená pro distribuované prostředí) a možnost kdekoliv spuštění vytvořených komponent. Ember získal svoji popularitu především díky používání modulu Fastboot.js, který zpracovává vykreslování DOM struktury stejně jako React, a zároveň využívá dvoucestné synchronizace dat, jako je tomu u Angularu. [4][7]

Meteor – je javascriptový framework vyvíjený společností Meteor Development Group. Jeho vydání bylo v roce 2012, jako open source framework pod MIT licencí. Meteor se od ostatních uvedených frameworků liší tím, že pro vývoj aplikace používá pouze čistý javascript. Aplikace je tedy celá, od začátku až do konce, napsaná v jednom jazyce a není potřeba se učit nic nového, co by bylo potřebné pro efektivní

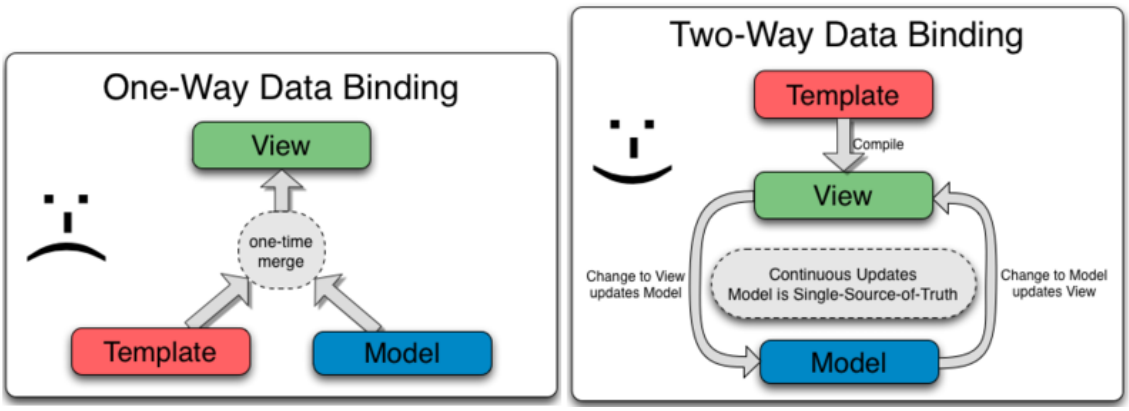
využití tohoto frameworku. Meteor disponuje potřebnými funkcemi pro vykreslování frontendu, rozvoj backendu či databází. Tento framework je modulární, takže je možné využít další knihovny, či vytvářet svoje vlastní komponenty, které na straně serveru lze spustit pomocí Node.js. Díky jednoduchosti a rychlosti vyvíjení aplikací, Meteor stále roste na popularitě mezi komunitou vývojářů a na internetu je k dohledání spousta výukových programů, rozšiřujících knihoven či komponent. [4][8]

1.2 Volba frameworku a databáze

Pro vývoj aplikace, kterým se tato práce zabývá, byl zvolen javascriptový framework Angular a databáze Caché od InterSystems. Při této volbě se hledělo na výhody tohoto frameworku s databází, a v neposlední řadě rozhodly i osobní preference.

Při volbě frameworku se rozhodovalo mezi Angularem a Reactem. Nakonec volbu vyhrál Angular, díky už předešlým zkušenostem s tímto frameworkem a jeho výhodami. Vyvíjená aplikace nebude složitá na práci s objektovým modelem dokumentu, proto zde není nutné použití Reactu, který by byl v opačném případě vhodnější volbou, pro jeho rychlejší vykreslování složitějších uživatelských rozhraní. Oproti tomu, Angular může nabídnout dvoucestný binding, kde veškerou práci s aktualizováním view vrstvy a vrstvy modelu řeší framework za nás, a proto nemusíme zasahovat vlastním kódem do objektového modelu dokumentu. Další výhodou Angularu je jeho cross-platformní využití, kde jde o to, že vytvořené komponenty můžeme využívat jak pro vývoj webové aplikace, která bude zobrazitelná na počítači a zároveň i na mobilním zařízení bez jakýchkoliv potřebných změn, tak i pro vývoj nativních mobilních aplikací nebo nativních desktopových aplikací. Za zmínku stojí i jeho pravidelný vývoj, kdy vývojáři Angularu přinesou vždy něco nového, společně se zlepšením již stávajících funkcí. Příkladem může být první verze Angularu – AngularJS, který se nesetkal s příliš velkou oblibou, především kvůli jeho pomalé rychlosti při webových aplikací. Tento problém se hned s druhou verzí, Angular2, odboural a Angular se stal jedním z nejrychlejších webových javascriptových frameworků, dnes

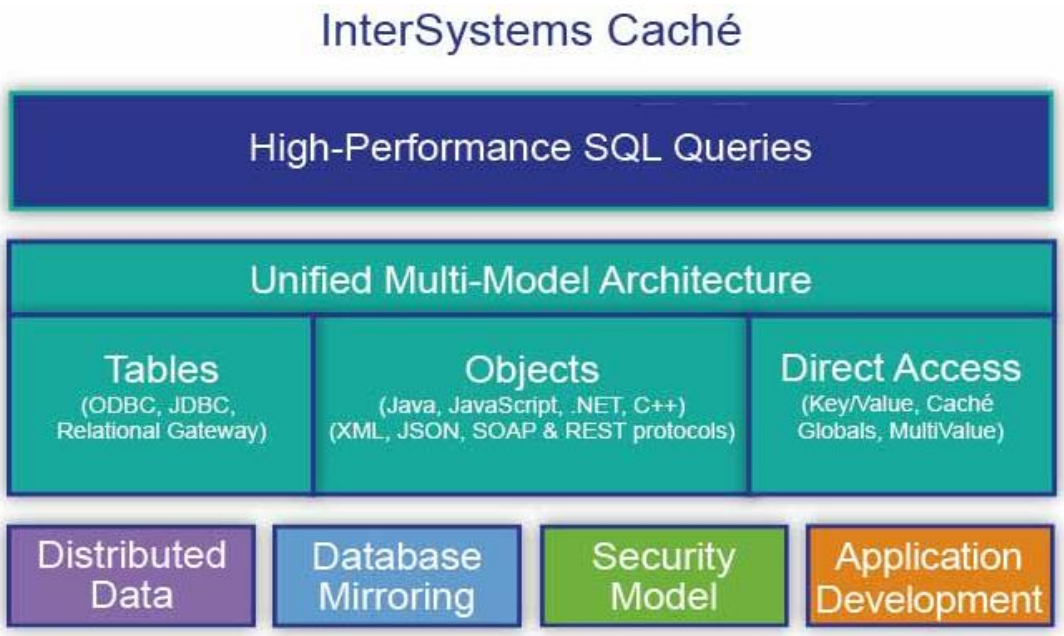
už máme čtvrtou verzi Angularu. Velkým plusem může být využití, pro psaní webové aplikace, editoru Visual Studio Code, které má plnou podporu Angularu a obsahuje nespočet nápověd při psaní a ladění aplikace. Další výhodou je jednoduchá návaznost Angularu na ostatní aplikace společnosti Google, pomocí již zabudovaných komponent je tak možné aplikaci spojit například s Google Maps, Youtube, či dalšími aplikacemi. V neposlední řadě má Angular velkou fanouškovskou základnu, díky ní lze na internetu dohledat nespočet návodů, řešených problémů, či již vytvořených komponent nebo pluginů. [5][9]



Obrázek 1 - Jednocestný versus dvoucestný binding [9]

Při volbě databázového systému se vycházelo ze závěrů ročníkové práce, která se, v neposlední řade, zabývala i volbou vhodné databáze pro aplikaci zákaznického portálu. Bylo zde vyjmenováno několik možných druhů databází se svými hlavními zástupci, z nichž jako nejvhodnější řešení se jevilo použití objektově orientované databáze Caché od společnosti InterSystems. Jedná se o velice pokročilý systém řízení báze dat a prostředí, který umožňuje zpracování a analýzu velkého objemu složitých dat a je vhodný pro vývoj webových a mobilních aplikací. Caché disponuje vysokým výkonem, masivní škálovatelností a robustní spolehlivostí. Uložená data jsou popsána v integrovaném slovníku dat a jsou nepřetržitě k dispozici prostřednictvím objektového přístupu, vysoce výkonného jazyka SQL či rychlého přístupu k vícerozměrným datovým strukturám. Všemi těmito způsoby lze přistupovat k datům současně, bez jakéhokoliv problému. Další výhodou této databázové platformy je

integrovaná technologie Zen, která umožňuje rychle vytvářet obsáhlé webové aplikace s krátkou dobou odezvy a svůj výstup poskytuje v HTML5, který může být rovnou implementován na straně aplikace. Pro vysokou rychlost a škálovatelnost databáze je možné využívat webové služby, formáty REST a JSON či další standarty. Efektivní práce s vícerozměrnými datovými strukturami je řízena datovým strojem Caché, který pracuje s „řídkými poli" - pokud se ptáme na neexistující hodnotu, databáze nám vrací hodnotu „0" a udržuje pouze tolik položek, kolik je nenulových prvků. Díky tomuto způsobu databázový systém potřebuje méně hardwaru než systém založený na relačních databázích. V neposlední řadě je databáze Caché vybavena funkcí zrcadlení databáze, která zaručuje vysokou dostupnost a rychlé zotavení systému při plánovaných i neplánovaných výpadcích. [10][11][12]



Obrázek 2 - Databázový systém InterSystems Caché [12]

2 Příprava pro vývoj aplikace

Pro vývoj webové aplikace, využívající javascriptový framework Angular a databázi InterSystems Caché, potřebujeme stáhnout a nainstalovat potřebný software.

Angular využívá pro svůj chod softwarové prostředí Node.js, jedná se o prostředí navržené pro vývoj vysoce škálovatelných internetových aplikací, které jsou psané v javascriptovém jazyce. Základem Node.js je javascriptový engine V8 od společnosti Chrome, který je rozšířený o událostmi řízený I/O model a „balíčkový" ekosystém NPM, který je největším ekosystémem open-source knihoven na světě. Jeho instalace je snadná, stačí si stáhnout poslední doporučenou verzi (na námi používaný operační systém) z webové stránky nodejs.org a tu poté nainstalovat. Pro následné psaní kódu aplikace se doporučuje použití programu Visual Studio Code, jenž se jedná o open-source editor kódu, který je odlehčenou verzí Visual Studia, pro operační systémy Windows, Linux a macOS. Prostředí tohoto editoru je přehledně rozděleno do čtyř částí, hlavní z nich je plocha pro editaci kódu, kde můžeme mít otevřené až tři různé soubory vedle sebe, další částí je boční lišta, která obsahuje průzkumník souborů a seznam otevřených souborů, třetí částí studia je stavový řádek obsahující informace o souboru a projektu, a poslední částí je lišta zobrazení, která nás informuje například o počtu souborů ke commitu, pakliže používáme Github či podobnou službu. Mezi další pomocníky patří paleta příkazů, podpora kódování, přehledná instalace a správa rozšiřujících pluginů. Visual Studio Code je zdarma ke stažení ze stránky code.visualstudio.com. [13][14]

Posledním potřebným softwarem je Ensemble od InterSystems. Jedná se o platformu, která umožňuje vývoj a propojení serverové části aplikace s databází InterSystems Caché. Ensemble nabízí rychlý a hladký vývoj propojitelných aplikací se zabudovanými funkcemi pro integraci, pomocí této jedné jediné platformě dokážeme nahradit rozdílné integrační technologie a zaměřit se tak více na rozvíjení potřeb zákazníků. Platformu Ensemble, společně s databází Caché, lze po zakoupení licence stáhnout z webových stránek intersystems.com. [15]

2.1 Vytvoření a nastavení projektu

Jestliže máme veškerý potřebný software nainstalovaný na počítači, kde budeme aplikaci vyvíjet, můžeme se přesunout k samotnému vytvoření projektu. Nejlepším možným způsobem, jak vytvořit a spravovat aplikaci, je použití nástroje Angular CLI. Jedná se rozhraní příkazového řádku, které nám umožňuje vytvořit projekt, přidávat soubory, testovat aplikaci, přidávat a aktualizovat komponenty a mnoho dalšího. Použitím tohoto nástroje, je založení projektu aplikace rychlé a snadné.

Ze všeho nejdřív je potřeba si vytvořit adresář, kde budou soubory aplikace umístěny. Po vytvoření tohoto adresáře si otevřeme příkazový řádem a pomocí příkazu *cd /cesta_k_adresáři* se do adresáře přepneme. Následně pomocí příkazu *npm install -g @angular/cli* nainstalujeme nástroj Angular CLI. Po úspěšném nainstalování stačí zadat příkaz *ng new web_aplikace* a v podadresáři web_aplikace se nám vytvoří nový projekt. Nově vytvořený projekt už obsahuje všechny potřebné soubory či nastavení a pomocí příkazu *cd /web_aplikace* a následně *ng serve*, aplikaci zapneme. Po zadání adresy *http://localhost:4200/* do webového prohlížeče se nám zobrazí aplikace. Adresu s portem, na které aplikace běží, je možné změnit v souboru enviroment.ts, který se nachází v podadresáři enviroments (web_aplikace/src/enviroments/enviroment.ts). Ukončení běhu aplikace proběhne v okamžiku, jakmile zavřeme příkazový řádek, kde jsme danou aplikaci spustili. Opětovný start aplikace provedeme znovu pomocí příkazového řádku, kde se přesuneme do adresáře aplikace a zadáme příkaz *ng serve*. [16]

2.2 Přidání komponent a pluginů

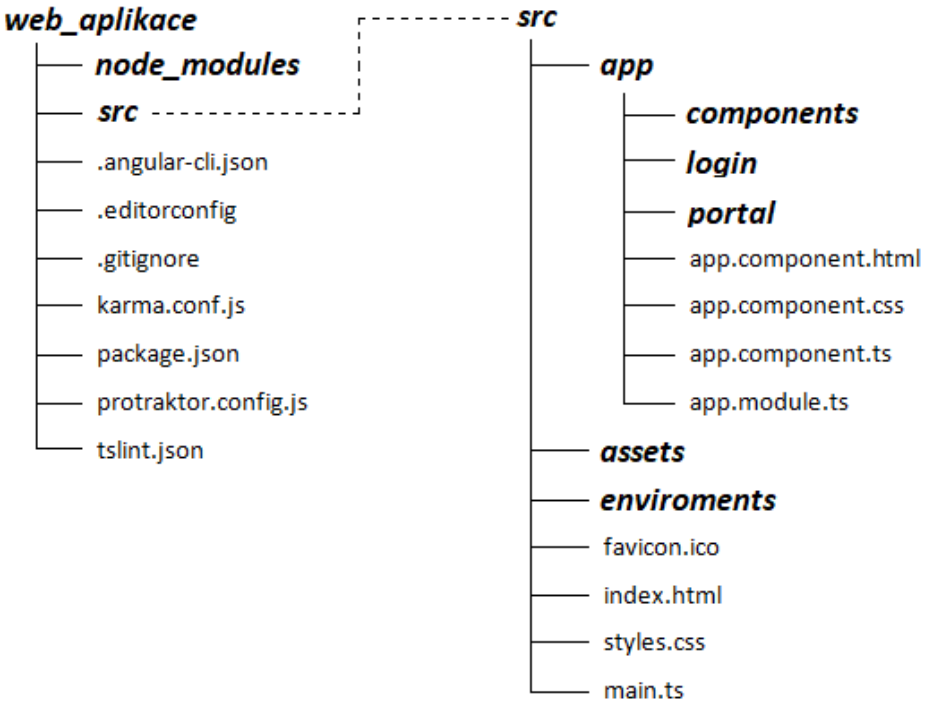
Jelikož je Angular jeden z nejrozšířenějších javascriptových frameworků na světě, spousta vývojářů vyvíjí svoje komponenty či pluginy a nabízí je volně ke stažení a nainstalování do projektu. Předvytvořené komponenty a pluginy nám usnadňují a urychlují programování aplikace, jako příklad můžeme uvést zabudování

kalendáře do aplikace, který nám bude umožnovat výběr dne pro zobrazování naměřených dat. Použitím komponenty, která kalendář obsahuje, si ušetříme čas při programování jeho funkcionality. Jednoduše kalendář pomocí speciálních tagů (např. tag *<p-calendar></p-calendar>* - kalendář z komponenty PrimeNG) vložíme do html stránky aplikace, dodefinujeme vstupy, výstupy, styl kalendáře a máme kalendář hotový. Samozřejmě takhle jednoduše to nejde u všech komponent či pluginů. Některé rozšíření a jejich funkce se musí doprogramovat tak, aby nám vyhovovali v našem použití. Při větších změnách je dokonce lepší předvytvořené komponenty či pluginy nepoužít a raději si naprogramovat vlastní.

Aplikace, kterou se tato práce zabývá, je rozšířena o čtyři hlavní komponenty a jeden plugin. První komponentou je Clarity Design System, kde se jedná o upravení vzhledu a rozšířenou funkcionalitu jednotlivých html tagů, jako jsou například tabulky, seznamy, různé bloky, odkazy a podobně. Další použitou komponentou je PrimeNG. Tato komponenta poskytuje podobné výhody jako Clarity Design System, s tím rozdílem, že zde nalezneme mapu, galerii či předvytvořený kalendář se zajímavými funkcemi. Pro zkrášlení aplikace pomocí ikon, je využita komponenta Font Awesome. Komponenta nabízí několik stovek profesionálně vytvořených ikon, které jsou připravené k použití do Angular projektů. Nalezneme zde jak ikony placené, tak i jejich druhé verze, které jsou zdarma. Největší výhodou těchto ikon je, že jsou ke stažení v svg formátu a vývojář si je může upravit podle svých představ. Poslední využívanou komponentou je Angular Google Maps. Jak už je z názvu patrné, je to komponenta potřebná pro integraci map od společnosti Google, které lze částečně upravovat. Tuto komponentu taktéž využívá, pro vykreslování map, komponenta PrimeNG. Vedle rozšiřujících komponent, je při vývoji využíván Sftp plugin pro Visual Studio Code, který zprostředkovává synchronizaci mezi lokálním a serverovým uložištěm, pro snadné stahování a nahrávání souborů. [17][18][19]

2.3 Struktura projektu a význam souborů

Po vytvoření projektu a nainstalování rozšiřujících komponent, pomocí nástroje Angular CLI, se v námi zadaném adresáři vygenerují soubory aplikace. Tyto soubory jsou uspořádány do přehledné struktury, podle toho, k čemu konkrétní soubor náleží a jaká je jeho úloha. Při rozšiřování aplikace, například vytvořením další aplikační stránky nebo jakýkoliv jiné komponenty, je doporučeno dodržovat danou strukturu projektu. Na níže uvedeném obrázku je zachycena struktura projektu, kterou tvoří jednotlivé soubory a adresáře aplikace (adresáře jsou zvýrazněny).



Obrázek 3 - Struktura projektu

V hlavním adresáři aplikace web_aplikace se nacházejí dva podadresáře a několik dalších souborů. V podadresáři node_modules se nacházejí soubory se zdrojovým kódem javascriptového enginu Node.js, který je potřeba pro běh aplikace vytvořené v Angularu. Podadresář src obsahuje soubory se zdrojovým kódem námi vytvářené aplikace. Toto je podadresář, kde aplikaci rozvíjíme a upravujeme, do jiné oblasti

není potřeba zasahovat, jelikož to za nás dělá nástroj Angular CLI. V souboru .angular-cli.json je konfigurace nástroje Angular CLI. Nalezneme zde informace o projektu, jako je název a verze, a konfiguraci samotné aplikace. Veškerá konfigurace, zde uvedená, je nastavovaná automaticky a není potřeba do ní zasahovat, výjimku tvoří nastavení cest ke stylům a skriptům, které komponenty aplikace využívají. Pokud chceme použít rozšiřující komponentu, kterou nelze automaticky nainstalovat pomocí nástroje Angular CLI, tak zde, v tomto souboru, ručně zadáme cesty ke skriptům a stylům přidávané komponenty a při znovuspuštění aplikace se komponenta načte. Soubor s názvem .editorconfig obsahuje konfiguraci pro editor kódu, v tomto případě pro Visual Studio Code. Pokud se rozhodneme využívat jiný editor, který soubor .editorconfig podporuje, tak se rovněž i u něho provede potřebné nastavení pro naši aplikaci. Pokud využíváme službu GitHub či podobnou, tak v souboru .gitignore můžeme nastavit cesty k souborům, které nechceme předávat pomocí příkazu „commit“, který adresář s aplikací, z lokálního počítače, odesílá do adresáře webové služby, kde je aplikace sdílená mezi více vývojářů, kteří mají k adresáři přístup. Soubor karma.conf.js obsahuje nastavení pro testovací nástroj Karma, který vytváří webový server, kde spouští zdrojový kód proti testovacímu kódu pro každý připojený prohlížeč. Výsledky tohoto testu jsou poté zanalyzovány a zobrazeny pomocí příkazového řádku vývojáři, který může na případné potíže s kompatibilitou reagovat úpravou kódu. V souboru package.json je konfigurace využívaných open-source knihoven z „balíčkového“ ekosystému NPM, který je součástí javascriptového enginu Node.js. U uvedeného názvu knihovny nalezneme označení nainstalované verze a informaci o tom, zda je dostupná novější verze knihovny, či námi využívaná verze je aktuální. Stejně jako u souboru karma.conf.js, představuje soubor proktraktor.config.js konfiguraci pro testovací nástroj Protractor, který je „end-to-end“ testovacím frameworkem pro Angular aplikace. Posledním souborem v hlavním adresáři aplikace je tslint.com, znova se jedná o konfiguraci, nyní pro nástroje TSLint a Codelyzer, které kontrolují a udržují zdrojový kód čitelný a bez chyb.

Nejdůležitější větví celé struktury projektu je podadresář src, kde se nachází soubory se zdrojovým kódem vyvíjené aplikace. Nachází se zde adresář app, který obsahuje jednotlivé části aplikace, jako jsou námi vytvářené komponenty, stránky aplikace a moduly. Je doporučené vytvářet, pro každou novou komponentu či

stránku aplikace, vlastní adresář a zařazovat ho do struktury projektu na místo, pod co spadá. Toto členění nám poskytuje jednoduchou a přehlednou orientaci v projektu, ale zároveň je tu zde i druhý důvod, který je mnohem závažnější a tím je udělování přístupu do jednotlivých částí projektu. Příkladem může být adresář login, který reprezentuje přihlašovací stránku do zákaznického portálu. Po načtení aplikace do webového prohlížeče, se uživateli zobrazí přihlašovací stránka, v tuto chvíli má uživatel přístup pouze v rámci adresáře login. Po úspěšném přihlášení ho směrovací služba aplikace přepne do adresáře portal, čímž získá přístup k jednotlivým částím zákaznického portálu. Tímto způsobem lze uživatele, v rámci aplikace, přemísťovat podle jejich pravomocí a zabránit tak nepovolenému přístupu. Každá komponenta či stránka aplikace má tři hlavní soubory se zdrojovým kódem. V případě aplikace jsou to soubory app.component.html, app.component.css a app.component.ts. První část názvu souboru (app) značí název komponenty, prostřední část (component) nám říká, že se jedná o komponentu a poslední částí je koncovka souboru. Pakliže má soubor koncovku html, tak se jedná o soubor s html zdrojovým kódem komponenty, soubor s koncovkou css obsahuje kaskádové styly komponenty a soubor s koncovkou ts obsahuje zdrojový kód komponenty napsaný v TypeScriptu. Dalšími soubory, které se mohou u komponenty vyskytovat, jsou app.module.ts a app.service.ts, taktéž se zdrojovým kódem psaným v TypeScriptu. Soubor app.module.ts obsahuje konfiguraci importovaných zdrojových kódů, deklarace a poskytovatele dané komponenty. Soubor app.service.ts je tzv. poskytovatel, kde je naprogramované například spojení mezi aplikací a databází. Dalším podadresářem v src je assets, zde se jedná pouze o adresář s obrázky či ikonami, které jsou využívané v aplikaci. Následuje podadresář enviroments, který obsahuje soubory enviroment.prod.ts a enviroment.ts, kde je zadaná url adresa aplikace a adresa, na které naslouchá databáze. Adresář src dále obsahuje soubory index.html, style.css a main.ts. Soubor index.html je hlavní html stránka aplikace, obsah této stránky se generuje automaticky nástrojem Angular CLI a není potřeba zde něco upravovat. Dalším souborem je style.css, jedná se o soubor s globálními kaskádovými styly, které jsou využívané napříč celou aplikací. Posledním souborem je main.ts, který je hlavním vstupním bodem aplikace provádějící kompilaci a zavádění kořenového modulu (app.module.ts) aplikace.

3 Vývoj aplikace

První problematikou, kterou se bude tato vývojová část práce zabývat, bude zprostředkování komunikace mezi aplikací a serverovou stranou s databází. Následně se práce bude zabývat vývojem samotných aplikačních stránek či jiných komponent, které se v zákaznickém portále vyskytují.

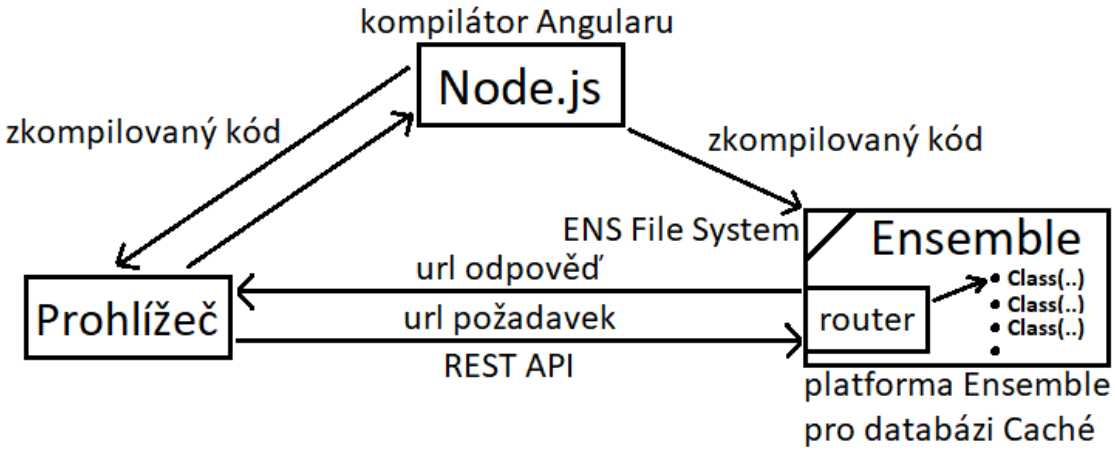
3.1 Komunikace mezi aplikací a databází

Zprostředkování komunikace mezi aplikací a databází, bylo jedním ze složitějších úkonů tohoto vývoje. Samotné provedení komunikace procházelo úpravami po celou dobu vývoje zákaznického portálu a vznikly dvě verze provedení, kdy nakonec byla využita verze druhá, čítající několik zásadních výhod.

3.1.1 První verze

V původní první verzi komunikace byly využívány celkem tři prvky. Prvním prvkem byl webový prohlížeč, následovalo softwarové prostředí Node.js a posledním prvkem byla platforma Ensemble na straně databáze. Spuštěný webový prohlížeč s aplikací zákaznického portálu využíval Node.js, který byl nainstalován na lokálním počítači, jako kompilátor Angularu při vývoji aplikace. Zkompilovaný kód se vracel zpět do prohlížeče a zároveň se ukládal i do platformy Ensemble. Jakmile byla v aplikaci vyvolána akce, která vyžadovala spojení s databází, tak se vytvořil url požadavek, využívající REST API, a odeslal se do Ensemble. V Ensemble se url požadavek rozbalil a podle doručených instrukcí se provedla požadovaná metoda na serverové straně s databází. Následně se výsledek metody odeslal, jako url odpověď, zpět do aplikace v prohlížeči, taktéž pomocí REST API.

Využívané REST API je veřejné rozhraní pro distribuované prostředí orientované na data, které umožňuje přístup k datům více uživatelům a aplikacím z rozdílných prostředí. V této práci využívá http protokol a je implementováno pomocí JSON formátu, který je velice rozšířený a podporovaný velkou škálou jazyků. [20][21]



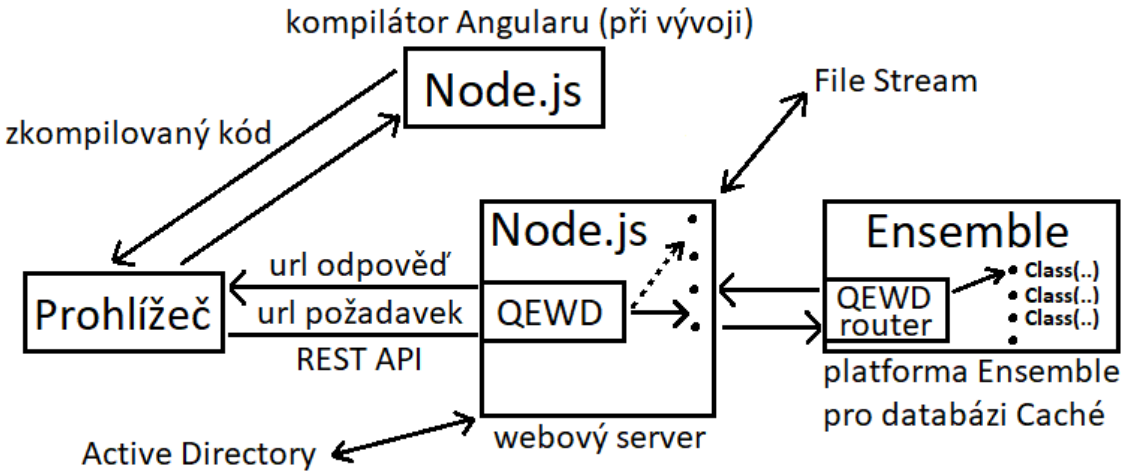
Obrázek 4 - Původní provedení komunikace aplikace s databází

Tato první verze komunikace, kde se přímo z aplikace dotazujeme databáze, je poměrně jednoduchá na provedení, avšak má pár nevýhod. Mezi nevýhody můžeme zmínit větší zatížení databáze, při dotazovaní několika stovek uživatelů ve stejný čas, nebo obtížné použití služeb Active Directory a FileStream. Z těchto důvodů byla počáteční verze komunikace předělána a vznikla verze druhá.

3.1.2 Druhá verze

V druhé verzi byl do komunikace, mezi prohlížečem a platformou Ensemble, přidán ještě jeden prvek. Nynější komunikace tedy čítá celkem čtyři prvky. Prvním prvkem je stále webový prohlížeč, kde běží aplikace zákaznického portálu. Druhý prvek, kterým je lokální Node.js, stále plní funkci kompilátoru zdrojového kódu Angularu a je potřeba po dobu vývoje aplikace. Po ukončení vývoje, aplikace poběží na webovém serveru a lokální Node.js už nebude potřeba. Třetím prvkem, který byl nově přidán,

je serverový Node.js, který běží na webovém serveru a posledním prvkem je platforma Ensemble na straně databáze.



Obrázek 5 - Finální provedení komunikace aplikace s databází

Změna nastala v komunikaci mezi prohlížečem a Ensemble, kde do tohoto informačního toku vstoupil webový server, na kterém běží serverový Node.js rozšířený o QEWD knihovnu. Zde se už nejedná o přímou komunikaci mezi aplikací a databází, ale po vyvolání akce v aplikaci, která vyžaduje zásah do databáze, se odešle url požadavek, pomocí REST API do QEWD knihovny. Tato knihovna požadavek zpracuje a v případě potřeby se vytvoří spojení mezi QEWD knihovnou a platformou Ensemble. V Ensemble se zavolá rutina QEWD Router, která obsahuje, z url požadavku, volanou metodu a objekt s parametry. Podle dotazované metody se vyhledá v Ensemble přidělená třída s metodou, kde se předají parametry a požadavek se zpracuje. Výsledná odpověď se poté odešle zpět do QEWD knihovny, kde je poté předána do aplikace v prohlížeči.

Použitím webového serveru, na kterém běží Node.js s knihovnou QEWD, jsme docílili několika výhod. První výhodou je řízení komunikace, které se řeší už na webovém serveru, QEWD knihovna pomocí routování sama řídí, na kterou třídu a metodu se má dotazovat a tím nahrazuje tuto práci na databázové straně. Jestliže počet dotazování vzrůstá, například připojením několika uživatelů ve stejnou dobu,

knihovna si sama vytvoří další komunikační uzly s databází a zvýší tak propustnost pro větší množství dotazů. Další velkou výhodou je vzájemné použití rozdílných databází. Knihovna QEWD může být připojena k několika databázím zároveň a pomocí výhybek řídit tok informací tam, kde je to potřeba. To dělá aplikaci, do jisté míry, nezávislou na typu databáze a současná databáze může být kdykoliv zaměněna, bez větších úprav. Další výhodou, která by byla v první verzi komunikace obtížně dosažitelná, je možnost používání Active Directory a File Stream. V případě Active Directory se jedná o adresářové služby LDAP implementované firmou Microsoft, tyto služby jsou rozšiřitelné a škálovatelné a umožňují efektivně uspořádat síťové prvky. Vedle informací o objektech v počítačové síti, poskytují také nastavování globálně systémové politiky, instalaci programů nebo aplikaci aktualizací v celé organizační struktuře. Vedle Active Directory, knihovna QEWD podporuje i File Stream, tudíž odpadá jakákoliv zdlouhavá práce s vytvářením funkcí pro stahování a nahrávání souborů. [22]

3.1.3 Realizace komunikace

V Angularu je komunikace realizována pomocí dvou rest metod, které se nacházejí v souboru app.service.ts a využívají http providera z Angularu. Tyto rest metody jsou volané z metod zákaznického portálu, nacházející se na jednotlivých stránkách nebo komponentách aplikace a z nichž přebírají parametry jako je cesta metody, volaná metoda a objekt s parametry.

První rest metoda, nazvaná jako RestQMethod, obsahuje tři parametry. Prvním parametrem je cesta k metodě (method) v QEWD knihovně, druhým parametrem je název volané metody (pFct) a posledním předávaným parametrem je objekt (pRequestObj), který obsahuje další parametry, jako je například identifikační číslo a heslo uživatele. Následuje vyplnění hlavičky, kde v Content-Type řekneme, že se jedná o formát JSON a přidáme token pro případné další ověření. Následně se provede POST akce HTTP protokolu se třemi parametry. První parametr obsahuje url adresu webového serveru, na který se dotazujeme, spojenou s cestou k metodě (url adresa serveru je definovaná v souboru enviroments.ts). Následuje parametr requestCover, který je naplněn názvem volané metody a objektem s parametry pro danou

metodu. Posledním parametrem je options, kde je obsažena vyplněná hlavička. Po provedení akce nám proměnou returnObj naplní dotazovaná data ve formátu JSON. Toto se provede pouze za předpokladu, že všechno proběhlo v pořádku, pokud někde nastala chyba aplikace nás pomocí this.router.navigate přesměruje zpět na přihlašovací stránku.

```
RestQMethod(method, pFct, pRequestObj = {}): Observable<any> {
        let headers = new Headers({
            'Content-Type': 'application/json',
            'authorization': 'Bearer ' + sessionStorage.getItem('token')
        });
        let options = new RequestOptions({ headers: headers });
        let requestCover = { 'fct': pFct, 'obj': pRequestObj };
        return this.http.post(`${this.url + method}`, requestCover,
options).map((response: Response) => {
                let returnObj = response.json();
                if (returnObj.status === -555) {
                    this.router.navigate(['login']);}
                if (returnObj.status === -2) { }
                return returnObj
            })}
```

Zdrojový kód 1 - Ukázka rest metody v Angularu

Druhou rest metodou je metoda nazvaná RestMethod. Tato metoda se od předchozí liší pouze v předávaných parametrech, které jsou zde pouze dva. Tato metoda je využívaná při přihlašování, tudíž se zde předává pouze cesta k metodě a objekt obsahující parametry identifikační číslo a heslo uživatele. V tomto případě není potřeba předávat název volané metody, protože QEWD knihovna podle cesty k metodě sama zjistí, o jakou metodu se jedná a komunikaci přesměruje rovnou na přihlašovací metodu v Ensamble. Po nainstalování Node.js s knihovnou QEWD na webovém serveru, je potřeba propojit aplikaci s databází. V první řade byla provedena konfigurace sftp pluginu, který je využíván editorem Visual Studio Code, pro usnadnění nahrávání souborů QEWD knihovny na webový server, pokud byly upraveny. Konfigurace je v souboru sftp.json a obsahuje ip adresu a port webového serveru, přihlašovací jméno a heslo, a cestu, kam se soubory knihovny QEWD ukládají. Následně byla provedena konfigurace spojení QEWD knihovny s databází, která se nachází v souboru

rest.js a obsahuje parametr jako název serveru, port, počet uzlů a údaje o databázi, jako je typ databáze, cesta k databázi, přihlašovací jméno s heslem a název jmenného prostoru, kde se nacházejí metody databáze. V souboru myRestService.js je poté nadefinováno co se má stát po zavolání metody z aplikace. Pokud do QEWD knihovny přijde požadavek ze zákaznického portálu na provedení rest metody, jejíž cesta je /api/loginCP, tak se v module.exports metodě přiřadí handler login a provede se funkce login, která obsahuje dva parametry. V argumentu args je uloženo identifikační číslo a heslo uživatele, který se chce přihlásit a v parametru finished se předává výsledek funkce. Na začátku funkce je potřeba ověřit, jestli předávané identifikační číslo či heslo není prázdné, v takovém případě by se vrátil výsledek funkce s upozorněním na vyplnění těchto údajů. Pokud identifikační číslo a heslo vyplněné je, vytvoří se spojení s QEWD na straně databáze obsahující parametry function, kde je zadaný název funkce, který se má provést a parametr arguments, který předá identifikační číslo a heslo uživatele. Pokud přihlášení proběhne, vrátí se objekt se statusem 1, tak se vytvoří session, která uchovává token, id a jméno uživatele. Vytvořený token se poté, společně s objektem, vrátí v parametru finished, jako výsledek funkce. Jestliže se vrátí objekt se statusem jiným, než je 1, tak se v parametru finished vrátí status objektu jako chyba s popisem.

```
function login(args, finished) {
  var identifier = args.req.body.identifier;
  if (!identifier || identifier === '') {
    return finished({error: 'You must provide a identifier'});}
  var password = args.req.body.password;
  if (!password || password === '') {
    return finished({error: 'You must provide a password'});}

  var loginResult = this.db.function({function:
      'login^qewdCpConsumer',arguments: [identifier,password]});
  let rObj=JSON.parse(loginResult)
  if (rObj.status==1) {
      var session = this.sessions.create('testWebService', 3600);
      session.authenticated = true;
      session.data.$('id').value = rObj.data.id;
      session.data.$('name').value = rObj.data.name;
```

Zdrojový kód 2 – Ukázka řízení komunikace na webovém serveru (funkce login)

Pokud do QEWD knihovny přijde požadavek na vykonání metody s cestou /api/cp, tak se metodě přiřadí handler serverF a provede se funkce serverF, která obsahuje taktéž parametry args a finished. Zde již neprobíhá žádná kontrola zadaných údajů, ale provede se rovnou spojení s QEWD na straně databáze, kam se znova předá název metody, která se má vykonat, a parametr arguments obsahující identifikační číslo uživatele nebo smlouvy (podle toho o jakou dotazovanou metodu se jedná), název dotazované metody a objekt s dalšími parametry.

```
function serverF(args, finished) {
   var fctResult = this.db.function({function:
'fctRouter^qewdCpConsumer', arguments: [args.session.id,args.req.path,
JSON.stringify(args.req.body)]});
  finished(fctResult.result);
}
```

Zdrojový kód 3 – Ukázka řízení komunikace na webovém serveru (funkce serverF)

Poslední částí je zrealizování komunikace v Ensemble na straně databáze. Zde bylo potřeba doprogramovat QEWD routování, které po zaznamenání požadavku, který přišel z QEWD knihovny na webovém serveru, zjistí, o jakou volanou metodu se jedná a nasměruje jí na místo, kde se poté metoda vykoná. Routování je definováno v Ensemble v souboru qewdCpConsumer.mac, kde se zavolá funkce fctRouter (v případě, že se jedná o přihlášení uživatele, se zavolá funkce login) obsahující tři parametry. Prvním parametrem je session, který obsahuje session předanou z QEWD knihovny webového serveru a pakliže session existuje, tak vytvoří novou, kde v Id bude nahrána stará session, v CustomerId je identifikační číslo uživatele a v CustomerName je jméno uživatele. Následuje parametr url, kde je uložená cesta k metodě. Pokud se cesta rovná /api/cp, tak je požadavek přesměrován do třídy CP.API.Portal, která má zděděné veškeré metody, které zákaznický portál využívá. Následně se z posledního parametru, kterým je request, zjistí, o jakou metodu jde a předají se jí potřebné parametry, které se nacházejí v objektu uvnitř parametru. Poté, co se požadovaná metoda provede, odešle se zpět odpověď, která v sobě nese status a data. Pokud vše proběhlo v pořádku, tak status odpovědi bude roven jedné a v data bude

objekt s dotazovanými daty. V opačném případě se vrátí odpověď se statusem nula, který poté zaregistruje QEWD knihovna na straně webového serveru a upozorní uživatele portálu chybovou hláškou.

```
fctRouter(session,url,request) {
      S $ZT="err"
      ;S ^tq.qewd("requestJSON")=request.%ToJSON()
      K %xSession,%xRequest, response
      if $L(session) {
      If $D(^CacheTempEWDSession("session",session)) {
      S %xSession={}
      S %xSession.Id=session
      S %xSession.CustomerId=$G(^CacheTempEWDSession
            ("session",session,"id"))
      S %xSession.CustomerName=$G(^CacheTempEWDSession
            ("session",session,"name"))
      }}
      Q:'$D(%xSession)
      S %xRequestCover={}.%FromJSON($ZCVT(request,"i","UTF8"))
      if url="/api/cp" S response=$$serve("CP.API.Portal")
      If '$D(response) S response=$$err1() /// pokud nedošlo ke shodě,
tak vrátíme náhradní response s chybou
      S responseJSON=$ZCVT(response.%ToJSON(),"o","UTF8")
      Q responseJSON
}
```

Zdrojový kód 4 – Ukázka řízení komunikace v Ensemble (neplatí pro přihlašování uživatele)

3.2 Přihlášení do zákaznického portálu

Při vytváření přihlašovací stránky do zákaznického portálu, bylo nejdříve potřeba vytvořit komponentu, která bude, přihlašovací stránku v aplikaci, reprezentovat. Komponentu můžeme vygenerujeme pomocí nástroje Angular CLI. Jednoduše se v příkazovém řádku přesuneme do adresáře, kde chceme vytvořit komponentu a zadáme příkaz *ng g component login*, po krátké chvilce je komponenta vygenerovaná a byla importovaná do všech potřebných modulů v aplikaci – můžeme přejít rovnou k upravování komponenty. Vytvoření komponenty můžeme provést taktéž i ručně, bez použití nástroje Angular CLI, kdy jednotlivé soubory komponenty vytvoříme a jejich url cesty zadáme do souboru app.module.ts.

U vytvořené komponenty, kde se jedná o stránku aplikace, je nejdůležitější provést nejdříve nadefinování modulu a routování stránky. V modulu stránky login.module.ts jsou importovány ostatní komponenty, moduly či service soubory, které přihlašovací stránka využívá. V tomto případě jsou zde importovány základní moduly z Angularu, modul rozšiřující komponenty Clarity Design System, komponenta reprezentující samotnou přihlašovací stránku (login-page) a service soubor app.service.ts, kde jsou definovány rest metody, pro komunikaci s databází (viz kapitola 4.1.3). V souboru login-routing.module.ts je definované routování přihlašovací stránky. Zde je uvedený pouze jeden záznam – cesta ke komponentě přihlašovací stránky. Pokud kdekoliv zadáme url adresu aplikace s /login nebo použijeme přesměrování na login, tak nás to přesune na komponentu LoginPageComponent a zobrazí se nám přihlašovací stránka zákaznického portálu.

```
//login.module.ts
…
@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    LoginRoutingModule,
    ClarityModule
  ],
  declarations: [LoginPageComponent],
  providers: [AppService]
})
export class LoginModule { }

//login-routing.module.ts
const routes: Routes = [
  { path: 'login', component: LoginPageComponent }
];
…
```

Zdrojový kód 5 – Ukázka modulu a routování přihlašovací stránky v Angularu

Pakliže máme routování a modul stránky nadefinovaný, můžeme se přesunout na vzhled a funkčnost samotné přihlašovací stránky. V podadresáři login-page jsou vytvořeny tři soubory. Prvním souborem je login-page.component.css, zde jsou nadefinované lokální kaskádové styly. V případě přihlašovací stránky jsou tu úpravy ve

vzhledu a umístění přihlašovacího tlačítka, jehož globální styl se nachází v style.css. Následuje soubor login-page.component.html, kde je definován html zdrojový kód stránky. Tato stránka využívá základní html tagy jako je například form, div, label, span, ale zároveň se zde vyskytují i tagy z rozšiřující komponenty Clarity Design System, jako je clr-main-container, který stanovuje, jak bude stránka rozvržená. Clarity Design System předává svoje stylovaní i na základní html tagy, které je poté ještě upravováno v souboru style.css, aby se docílilo požadovaného vzhledu stránky. Funkční část stránky se nachází v souboru login-page.component.ts, kde jsou pomocí TypeScriptu napsané třídy a metody. Přihlašovací stránka využívá pouze jednu metodu a tou je doLogin. Tato metoda je vyvolaná po stisku přihlašovacího tlačítka a její úlohou je zprostředkování přihlášení uživatele do zákaznického portálu.

```
async doLogin() {
    try {
      this.isLogging = true;
      this.LoggingError = false;
      this.appService.RestMethod('/api/loginCP', { 'identifier':
this.identifier, 'password': this.password }).subscribe(result => {
        if (result.rObj) {
          this.token = result.token;
          var customer = result.rObj.data;
          if (this.token) {
            sessionStorage.setItem('token', this.token);
            sessionStorage.setItem('id', customer.id);
            sessionStorage.setItem('name', customer.name);
            this.isLogging = false;
            this.router.navigate(['portal']);
          } else {
            this.isLogging = false;}
        } else {
          this.isLogging = false;
          this.LoggingError = true;}
      })}
```

Zdrojový kód 6 - Přihlašovací metoda v Angularu

Po vyvolání doLogin metody se zavolá rest metoda RestMethod, která se nachází v app.service.ts a předají se jí dva parametry. Prvním parametrem je cesta k volané metodě a druhým parametrem je objekt s identifikačním číslem a heslem uživatele. Rest metoda vytvoří http požadavek a odešle ho do QEWD knihovny na webovém

serveru, zde se podle zadané cesty k metodě (/api/loginCP) přiřadí k požadavku handler login a provede se funkce login. Zde se ověří, jestli předávaný parametr objektu s identifikačním číslem a heslem uživatele není prázdný. Pokud je vše v pořádku vytvoří se spojení s QEWD knihovnou na straně databáze, kde se provede funkce login, která přesměruje požadavek na přihlašovací metodu QewdLogin, vyskytující se v třídě CP.API.Secure.Login v Ensemble.

```
login(identifier,password)
      K ^tq.qewd
      S ^tq.qewd(0)="login"
      S response=
##class(CP.API.Secure.Login).QewdLogin(identifier,password)
      S ^tq.qewd(1)=$LB(identifier,password)
      S responseJSON=$ZCVT(response.%ToJSON(),"o","UTF8")
      S ^tq.qewd(2)=responseJSON
      Q responseJSON
```

Zdrojový kód 7 - Řízení komunikace pro přihlašování uživatele v Ensemble

V metodě QewdLogin, která obsahuje parametry pIdentifier a pPassword (identifikační číslo a heslo zákazníka) se nejdříve provede metoda AuthorizeCustomer, kde se ověří, jestli daný zákazník existuje. Pokud ano, tak se přejde k ověření hesla. Pakliže vše proběhne v pořádku, vrátí se autorizace s hodnotou jedna. V odpovědi se přidělí statusu hodnota jedna a do dat se přidělí jméno a id uživatele, které je poté využívané, jako jeden z parametrů při volání ostatních metod. Jestliže autorizace neproběhne, ať už v důsledku nenalezení uživatele nebo špatně zadaného hesla, vrátí se odpověď se statusem 0, který při návratu zachytí QEWD knihovna a vypíše chybu, která se zobrazí uživateli na přihlašovací stránce. V prvním případě, kdy autorizace proběhla, QEWD knihovna vytvoří token a vrátí ho zpět, i s odpovědí z databáze, přihlašovací metodě doLogin v Angularu. Ta získaný token, id uživatele a jméno uživatele uloží do session a přesune uživatele, pomocí routování, do zákaznického portálu. Uživatel byl přihlášen a může využívat poskytované služby zákaznického portálu.

```
ClassMethod QewdLogin(pIdentifier, pPassword)
{
	S oResponse=..GetResponse()
	S authorize=..AuthorizeCustomer(pIdentifier,pPassword,.oCustomer)
	if (authorize=1) {
		S oResponse.data.id=oCustomer.%Id()
		S oResponse.data.name=oCustomer.Name
		S oResponse.status=1
	} else {
		S oResponse.data.user=""
		S oResponse.data.name=""
		S oResponse.status=0
	}
	Q oResponse
}

ClassMethod AuthorizeCustomer(pIdentifier, pPassword, ByRef oCustomer)
As %String
{
	S oCustomer=##class(Ap.NEO.D.Customer)
		.IndexIdentifierOpen(pIdentifier)
	If $IsObject(oCustomer) {
		If oCustomer.CpPasswd=pPassword {
			Q 1
		} else { Q -1 }
	} else { Q -2 }
}
```

Zdrojový kód 8 - Přihlašovací a autorizační metoda v Ensemble

Výsledný vzhled přihlašovací stránky zákaznického portálu je možné si prohlédnout v příloze A Obrázky aplikace pro zákazníky distribuce elektrické energie, kde jsou k nahlédnutí obrázky se vzhledem všech částí a stránek zákaznického portálu.

3.3 Menu a automatické odhlašování

Po přihlášení do aplikace, se uživateli zobrazí stránka s výběrem smluv a v horní části okna se nachází jednoduché menu, pomocí něhož se uživatel dostane na podrobnosti zákaznického účtu, výpis faktur nebo zpět na výpis smluv. V souborech, kde je menu definované, se vyskytuje i zdrojový kód automatického odhlašovaní, které po určité době neaktivity v zákaznickém portálu, upozorní uživatele vypsáním

varování, že bude automaticky odhlášen. Jakmile i přes upozornění nenastane aktivita, uživatel je do pár sekund odhlášen a je zpět přesměrován pomocí routování na přihlašovací stránku.

Zdrojový kód menu a automatického odhlašování se nachází v adresáři portal, kde je podadresář layout se soubory layout.component.css, layout.component.html a layout.component.ts. Menu je vytvořené v souboru layout.component.html, pomocí html tagů, které jsou modifikované komponentou Clarity Design System. Pomocí tagu clr-main-container je definovaná zobrazovací oblast aplikace. Tato oblast je rozdělená na dvě části, první část reprezentuje tag clr-header, který je hlavičkou zobrazovací oblasti a je zde zobrazen název aplikace, následují tři odkazy, které mají vzhled ikon a plní funkci menu. V pravé části hlavičky je zobrazeno jméno přihlášeného uživatele (zákazníka) a možnost pro odhlášení. Druhou částí zobrazovací oblasti je oblast pro jednotlivé stránky zákaznického portálu a je definována pomocí tagu router-outlet. Při přechodu, pomocí menu, na jinou stránku se tedy nepřepisuje html kód celé aplikace, ale přepíše se pouze obsah v router-outlet – mění se pouze oblast pod menu. Ve stejném souboru se nachází i html zdrojový kód varování zobrazované při neaktivitě. Toto varování se zobrazuje v pruhu po celé šířce aplikačního okna nad oblastí s menu a obsahuje text s upozorněním, počet sekund do odhlášení a tlačítko, které po stisku vyvolá metodu reset, která zruší odpočítávání do odhlášení uživatele. Soubor layout.component.css opět slouží pro definování lokálních kaskádových stylů, které v tomto případě nejsou využívány, využíván je pouze předdefinovaný styl z komponenty Clarity Design System a mírné úpravy provedené pomocí globálních kaskádových stylů definovaných v souboru style.css.

```
<clr-main-container>
    <clr-header class="header-4">
        <div class="branding">
            <div class="nav-link"><i class="fab fa-accusoft fa-2x"></i>
                <span style="margin-left:10px" class="title">Zákaznický
portál</span></div>
        </div>
        <div class="header-nav">
            <a class=" nav-link nav-icon" routerLink='/portal/contracts'
routerLinkActive='active'>
                <i style="left: 50%; top: 50%; position: absolute;
transform: translate(-50%, -50%)" class="fas fa-th-list fa-lg"></i></a>
            …
        </div>
        <div class="header-actions">
            <div id="user-header">{{ Name }}</div>
            <a class="nav-link nav-text" (click)="logout()"
style="cursor:pointer">
                <i style="left: 50%; top: 50%; position: absolute;
transform: translate(-50%, -50%)" class="fas fa-sign-out-alt fa-lg"></i>
            </a>
        </div>
    </clr-header>
    <div class="content-container">
        <div class="content-area"><router-outlet></router-outlet></div>
    </div>
</clr-main-container>
```

Zdrojový kód 9 – Část html kódu menu zákaznického portálu v Angularu

V souboru layout.component.ts jsou naprogramované metody pro odhlášení, řízení automatického odhlášení a jeho resetování. Metoda logout je vyvolávána tlačítkem, které slouží pro ruční odhlášení. Po zavolání metody se ze session odebere token, jméno a id uživatele a pomocí routování se uživatel přesměruje na přihlašovací stránku. Automatické odhlašování je definované v konstruktoru třídy komponenty a využívá importované moduly Idle a DEFAULT_INTERRUPTSOURCES z Angularu. Pomocí metody idle.setIdle se nastaví délka neaktivity v sekundách, po jejíž uběhnutí se zobrazí uživateli varovaní. Délka odpočítávání do odhlášení uživatele, která je zobrazována v upozornění, je zadána pomocí metody idle.setTimeout. Metodou idle.setInterrupts s parametrem DEFAULT_INTERRUPTSOURCES, je nastaveno přerušení automatického odhlášení po kliknutí, tažení či pohybování myší po stránce aplikace. Jakmile nastane některá akce z uvedených možností přerušení, tak se zavolá metoda idle.onIdleEnd, která odpočítávání přeruší a zakáže zobrazování

varování. Jestliže odpočítávání do odhlášení vyprší, provede se metoda idle.onTimeout, která přeruší odpočítávání, zakáže zobrazování upozorňování a přesune uživatele pomocí routování na přihlašovací stránku – uživatel byl odhlášen. Metoda idle.onIdleStart a idle.onTimeoutWarning jsou vyvolány při zaznamenání neaktivity a provedou zobrazení varování a start odpočítávání do odhlášení.

```
constructor(private router: Router, private appService: AppService,
private idle: Idle) {
    idle.setIdle(300);
    idle.setTimeout(30);
    idle.setInterrupts(DEFAULT_INTERRUPTSOURCES);
    idle.onIdleEnd.subscribe(() => {
        this.timedOut = false;
        this.timeoutAlertShow=false;
    });
    idle.onTimeout.subscribe(() => {
        this.timedOut = false;
        this.timeoutAlertShow=false;
        this.router.navigate(['login']);
    });
    idle.onIdleStart.subscribe(() => {
        this.timedOut = true;
        this.timeoutAlertShow=true;
    });
    idle.onTimeoutWarning.subscribe((countdown) => {
        this.idleState = 'Budete odhlášeni za ' + countdown + '
sekund!';
        …
    });
    this.reset();}
```

Zdrojový kód 10 - Část kódu automatického odhlašování v Angularu

3.4 Výpis všech smluv zákazníka

Seznam všech smluv zákazníka je zobrazován na úvodní stránce, která se zobrazí ihned po přihlášení do zákaznického portálu. Jednotlivé smlouvy se zobrazují v kartách, které nesou informace o čísle smlouvy, platnosti, obchodníkovi s energií, distribučním regionu a jestli je smlouva aktivní. Po kliknutí na kartu se zavolá metoda,

které nás přesměruje na detail dané smlouvy, kde jsou k vidění podrobnější informace, naměřené údaje a graf.

Zdrojový kód stránky, se seznamem smluv, se nachází v podadresáři contracts v souborech komponenty contracts.component.css, contracts.component.html a contracts.component.ts. V contracts.component.css je provedená menší lokální úprava vzhledu karet a nadpisů, pomocí kaskádových stylů. V souboru contracts.component.html se nachází html zdrojový kód stránky. Uspořádání stránky a výpis jednotlivých karet je znovu řešený pomocí html tagů využívající rozšiřující komponentu Clarity Design System, která upravuje jejich vzhled. Samotný výpis karet je řešen pomocí html tagu div, který má přidělenou metodu *ngFor z Angularu. Tato metoda provede cyklus, který vygeneruje tolikrát html tag div, kolik je položek v přiděleném poli. Uvnitř vygenerovaných divů se poté vyskytuje html kód karty, která obsahuje výše zmíněné informace o smlouvě. Pro zobrazení informace, jestli je smlouva aktivní či neaktivní, je použita metoda *ngIf, taktéž metoda z Angularu, která zobrazí vnořený obsah s informací o stavu, pouze za předpokladu, pokud se obsah proměnné, kde je uložen stav smlouvy, rovná s uvedenou hodnotou. Jedná se tedy o metodu, provádějící podmínku If.

V souboru contracts.component.ts se nacházejí metody, které stránka využívá. První metodou je GetContracts s parametrem Id uživatele, který se pomocí konstruktoru třídy vezme ze session, kam se Id uživatele po přihlášení uložilo. Metoda GetContracts je automaticky zavolaná při inicializaci stránky a provede spojení s QEWD knihovnou na straně webového serveru pomocí rest metody RestQMethod, která se nachází v app.service.ts, a předá jí tří parametry. Prvním je cesta k metodě, druhým parametrem je název volané metody a posledním parametrem je objekt obsahující identifikační číslo uživatele.

```
export class ContractsComponent implements OnInit {
  public Contracts:any=[];
  Id:string;

  public GetContracts(Id) {    this.appService.RestQMet-
hod('/api/cp','GetCustomerContractsCP',{'id':Id}).subscribe(response=>
{
         this.Contracts=response.items;
    })}

  public GetContract(ContractId) {
       this.router.navigate(['/portal/contract/', ContractId]);}

  constructor(private router:Router, private appService:AppService) {
    this.Id = sessionStorage.getItem('id');}

  ngOnInit() {
    this.GetContracts(this.Id);}
}
```

Zdrojový kód 11 - Část kódu třídy s metodami stránky pro výpis smluv v Angularu

Podle prvního parametru, kde je zadaná cesta k metodě /api/cp, přidělí QEWD knihovna požadavku handler serverF a provede stejnojmennou funkci, která vykoná spojení s QEWD knihovnou na straně databáze. Zde se následně zavolá metoda GetCustomerContractsCP. Metoda nejprve vytvoří sql příkaz, který vybere všechna identifikační čísla smluv nacházející se v Ap_NEO_D_Customer_E.Contracts, u kterých se nachází zákazník s identifikačním číslem, které bylo předáno v posledním parametru požadavku. Následně se výsledek sql příkazu uloží do proměnné rs. V dalším kroku se provede cyklus while, kde pro každý záznam v proměnné rs, se pomocí uvedeného čísla smlouvy otevře přidělená smlouva s konfigurací cp_contracts, která nám vrátí požadované informace (stav smlouvy, číslo smlouvy, platnost, obchodník s energií, distribuční region). Získané informace se uloží do pole pxList, které se poté předá do parametru items odpovědi a odešle se zpět do QEWD knihovny. Ta odpověď předá zpět do aplikace zákaznického portálu, kde se z parametru items odpovědi vytvoří nové pole Contracts se smlouvami, které jsou následně vykresleny na stránce pomocí dříve zmiňované metody *ngFor v html kódu stránky.

```
ClassMethod GetCustomerContractsCP(req = "") As %Library.DynamicObject
{
	S oRequest=..GetRequest(req)
	S oResponse=..GetResponse()
	S pxList=[]
	S rs=##class(%Library.ResultSet).%New()
	S sql="SELECT ID From Ap_NEO_D_Customer_E.Contract WHERE
Customer="_oRequest.id
	D rs.Prepare(sql)
	D rs.Execute()
	While rs.Next() {
		S oPx={}
		S oPx.ContractId=rs.Get("ID")
	S oContract=
##class(Ap.NEO.D.Customer.E.Contract).%OpenId(rs.Get("ID"))
		S oPx.Contract=oContract.GetProxyObj(1,"cp_contracts")
		D pxList.%Push(oPx)
	}
	S oResponse.items=pxList
	Q oResponse
}
```

Zdrojový kód 12 – Metoda pro získání všech smluv zákazníka v Ensemble

3.5 Zákaznický účet uživatele

Pomocí volby v menu, se může uživatel dostat na svůj zákaznický účet, kde nalezne obecné informace o účtu, údaje v obchodním rejstříku, kontakty a potřebné adresy. Stránka je definovaná v podadresáři user, kde se nacházejí soubory user.component.css, user.component.ts a user.component.html se zdrojovým kódem. Vedle kontroly svých údajů o účtu, má zákazník také možnost, podat žádost na změnu některých údajů, jako je heslo, bankovní spojení, kontakty a podobně.

V souboru user.component.css jsou uvedeny lokální změny v kaskádových stylech, které jsou použity na nadpisy a sloupce formující zobrazované údaje. Html kód stránky je definovaný v souboru user.component.html. Vhled stránky je opět tvořen směsicí html tagů využívající styly a funkcionalitu z komponenty Clarity Design System. Jednotlivé části zobrazovaných údajů, jsou vybírány pomocí bočního menu – po kliknutí na položku v menu se přidělí jeho číselná hodnota do proměnné selectedMenu a následně se pomocí metody *ngIf ověří, který obsah náleží výběru a ten se

zobrazí. V horní části stránky se nachází tlačítko pro podání žádosti na změnu údajů, které po kliknutí vyvolá metodu OpenCustomModal. Ta vzápětí otevře na stránce modální okno a nabídne uživateli možnosti pro změnu jednotlivých údajů. Po vybrání údajů, které se budou měnit a jejich vyplnění, je uživatel, před odesláním žádosti, nucen zadat svoje aktuální heslo. Po vyplnění hesla a odeslání požadavku, je heslo serverem zkontrolováno a pokud se shoduje, požadavek je uložen do databáze a čeká na schválení, v opačném případě je uživatel upozorněn, že zadal špatné heslo a požadavek se neodešle. V souboru user.component.ts jsou definovány metody, které jsou stránkou využívány. Nalezneme zde metodu GetContact s parametrem ContactId, což je identifikační číslo zákazníka, které bylo pomocí konstruktoru třídy získáno ze session. Tato metoda je volaná pří inicializaci stránky a jako v případě získání smluv zákazníka, využívá rest metodu pro spojení s databází přes QEWD knihovnu. Po odeslání požadavku, na získání údajů o zákazníkovi, se zavolá metoda GetContactCP na straně databáze, která podle předaného parametru, který obsahuje id uživatele, otevře přiděleného zákazníka a vrátí konfiguraci cp. Ta obsahuje veškeré informace o zákazníkovi, které jsou v databázi uloženy. Získaná data se vrátí odpovědí zpět do zákaznického portálu, kde jsou poté uloženy do proměnné Contact, z které jsou následně vypisovány pomocí funkce ngModel do políček na stránce.

```
ClassMethod GetContactCP() As %Library.DynamicObject
{
	S oRequest=..GetRequest()
	S oResponse=..GetResponse()
	S oContact=##class(Ap.NEO.D.Customer).%OpenId(oRequest.id)
	S oResponse.data=oContact.GetProxyObj(1,"cp")
	Q oResponse
}
```

Zdrojový kód 13 – Metoda pro získání informací o zákazníkovi v Ensemble

Mezi další metody, nacházející se v souboru user.component.ts, patří metoda OpenCustomerModal, která se provede po stisknutí tlačítka, umožňující otevření modálního okna s výběrem údajů pro změnu. Tato metoda otevře modální okno, zavře rozevřené bloky reprezentující jednotlivé části údajů, které je možné změnit a zavolá rest metodu pro získání údajů z databáze, které jsou určené ke změně.

Odpovědí se vrátí dotazované údaje se stávající hodnotou a vypíšou se uživateli do modálního okna. Uživatel vybrané data přepíše podle potřeby a po zadání hesla, sloužící k ověření, odešle požadavek na změnu dat do databáze, kde se zavolá metoda ChangeRequestContactCP.

```
ClassMethod ChangeRequestContactCP(req = "") As %Library.DynamicObject
{
      S oRequest=..GetRequest(req) S oResponse=..GetResponse()
      S oCustomer=
##class(Ap.NEO.D.Customer).IndexIdentifierOpen(oRequest.login)
      If $IsObject(oCustomer) {
            If oCustomer.CpPasswd=oRequest.password {
                  S ChangeType = oRequest.changeType
                  S wtsDate = $$$Now
                  S date = $$$GetTime(wtsDate,"4.m")
                  If (ChangeType = "Password")
                  { S ^tq.ContactChanges(date,"Password") =
oRequest.data.%ToJSON() }
                  If (ChangeType = "Banking")
                  { S ^tq.ContactChanges(date,"Banking") =
oRequest.data.%ToJSON() }
                  If (ChangeType = "Contacts")
                  { S ^tq.ContactChanges(date,"Contacts") =
oRequest.data.%ToJSON() }
                  …
                  S oResponse.data.status = 1
                  S oResponse.data.result = "Váš požadavek byl přijat
do zpracování." } Else {
                  S oResponse.data.result = "Zadané heslo je chybné!"
                  S oResponse.data.status = 0
      …
```

Zdrojový kód 14 – Metoda pro podání žádosti o změnu údajů v Ensemble

Metoda ChangeRequestContactCP v první řadě provede ověření, jestli uživatel, který podává žádost existuje v databázi a jestli zadané heslo je správné. Pokud uživatel v databázi neexistuje vrátí se odpověď se statusem -1 a uživateli se vypíše varování na neexistujícího uživatele. Pokud uživatel existuje, ale bylo zadáno špatné heslo, vrátí se odpověď se statusem 0 a uživateli se vypíše varování, že zadal chybné heslo. Ani v jednom, z těchto dvou případů, se žádost o změnu údajů neprovede a neuloží do databáze. V opačném případě, kdy byl uživatel ověřen a heslo se shoduje, tak se

v databázi vytvoří globální proměnná, která má v parametru date nahraný datum, kdy byla žádost podána a v druhým parametru jsou uloženy údaje s novou hodnotou. Název druhého parametru globální proměnné se liší podle toho, o kterou část měněných údajů bylo požádáno, to zaručuje přehlednost v požadavcích na změnu údajů, které čekají na schválení.

3.6 Výpis všech faktur

Další možností zákaznického portálu je výpis faktur ze všech smluv vázající se na daného zákazníka. Tato stránka obsahuje jednoduchou tabulku, v níž jsou faktury reprezentovány pomocí řádků, které nesou základní informace, jako je název faktury, číslo smlouvy, ke které patří, účtovací období a stav počítadla. Po kliknutí na řádek v tabulce, se otevře pdf soubor faktury v nové kartě prohlížeče. Součástí stránky je rovněž i jednoduché filtrování faktur, podle zvolené smlouvy. Při inicializaci stránky je filtr nastaven na výchozí hodnotu, kdy vypíše faktury patřící ke všem smlouvám, pakliže si chce uživatel zobrazit pouze jednu konkrétní smlouvu, vybere si jí pomocí filtru a vypsané faktury se v tabulce zaktualizují.

Zdrojové kódy stránky se nacházejí v podadresáři invoices v souborech invoices.component.html, invoices.component.css a invoices.component.ts. Soubor invoices.component.html obsahuje html kód stránky, kde nalezneme html tagy table a div, formující vzhled stránky a tag clr-datagrid, který je z rozšiřující komponenty Clarity Design System a reprezentuje tabulku, do níž jsou faktury vypisovány. Jednotlivé řádky tabulky (datagridu) reprezentované tagem clr-dg-row jsou vypisovány funkcí *ngFor z pole faktur. V souboru invoices.component.ts nalezneme zdrojový kód třídy a metod stránky. Metoda GetInvoices s parametrem Id je volána při inicializaci stránky a proměnná Id je naplněná pomocí konstruktoru identifikačním číslem zákazníka ze session. Po zavolání metody se zavolá rest metoda, která se přes QEWD knihovnu spojí s databází, kde se provede metoda GetAllInvoices, nacházející se v třídě CP.API.Portal.

```
ClassMethod GetAllInvoicesCP(req = "") As %Library.DynamicObject
{
	…
	While rs.Next() {
		S oInvoicePx={}
		S ContractPx = {}  S oContract=
##class(Ap.NEO.D.Customer.E.Contract).%OpenId(rs.Get("ID")
		S ContractPx.id = rs.Get("ID")
		S ContractPx.value = oContract.Identifier
		D contractsList.%Push(ContractPx)
	S rs2=##class(%Library.ResultSet).%New("Ap.NEO.D.Acc.E.Invoice:
ContractInvoices")
		D rs2.Execute(rs.Get("ID"))
		While rs2.Next() {
			S oInvoicePx.id=rs2.Get("ID")
			S DateFrom=rs2.Get("DateFrom")
			S DateTo=rs2.Get("DateTo")
			S oInvoicePx.Contract = oContract.Identifier
			S oInvoicePx.Month = rs2.Get("Month")
			S oInvoicePx.DateFrom = $$$GetTime(DateFrom,"4.d")
			S oInvoicePx.DateTo = $$$GetTime(DateTo,"4.d")
	S oInvoicePx."Output_RealConsumption"=
rs2.Get("Output_RealConsumption")
			D pxList.%Push(oInvoicePx)
		} }
…
```

Zdrojový kód 15 – Metoda pro získání faktur ze všech smluv vázající se na daného zákazníka v Ensemble

Při provádění metody se nejdříve pomocí sql příkazu získají identifikační čísla všech smluv patřící zákazníkovi, jehož identifikační číslo se získalo z předaného parametru. Výsledek sql příkazu je uložen do proměnné rs. V následujícím while cyklu se pro každý záznam v rs proměnné provede několik příkazů. Nejdříve se pomocí identifikačního čísla otevře příslušná smlouva, z které získáme její název a provede se další while cyklus, který nalezne faktury přidělené k dané smlouvě. Ze záznamů o faktuře se vyberou data, jako je název faktury, fakturační období a stav počítadla, které se poté zapíší do pole pxList. Jakmile všechny cykly doběhnou a pole pxList je naplněné, tak se předá do parametru items v odpovědi a odpověď se odešle zpět přes QEWD knihovnu do zákaznického portálu. Zde se data z odpovědi uloží do pole Invoices, z kterého jsou následně faktury vypsány do tabulky. Data o názvu smluv se uloží do pole Contracts, jehož obsah je vypisován v selectu pro filtrování faktur podle určité smlouvy.

Při použití filtrování faktur podle určité smlouvy se zavolá metoda UpdateInvoices, která odkazuje na metodu GetInvoicesCP na straně databáze. Metoda GetInvoicesCP je obdobná předešlé metodě GetAllInvoicesCP s tím rozdílem, že se už neprovádí zjišťování faktur ze všech smluv patřících zákazníkovi, ale provede se vypsání faktur patřících pouze smlouvě, která byla ve filtru vybrána a jejíž identifikační číslo bylo metodě předáno pomocí parametru. Po nalezení faktur patřících smlouvě, metoda opět odešle odpověď s daty, které se znova nahrají do pole Invoices a tabulka s fakturami se zaktualizuje.

3.7 Hlavní dashboard smlouvy

Poslední a nejdůležitější částí zákaznického portálu je stránka s hlavním dashboardem smlouvy, který nám poskytuje detailní informace o konkrétní smlouvě, o přiděleném odběrným místě a v tabulce nám zobrazuje naměřené hodnoty z elektroměru pro vybraný den, které jsou zároveň promítnuty do grafu. Na tuto stránku se uživatel dostane po kliknutí (vybrání) na některou z karet, reprezentující jednotlivé smlouvy na stránce s výpisem všech smluv zákazníka.

Zdrojový kód stránky se nachází v podadresáři contract v souborech contract.component.html, contract.component.css a contract.component.ts. V souboru contract.component.css se opět nachází lokální úprava kaskádových stylu. V tomto případě jsou zde lehce upravené styly kalendáře z komponenty PrimeNG, seznamu elektroměrů a tabulky pro rezervovanou kapacitu. Soubor contract.component.html obsahuje html zdrojový kód stránky, obsahující několik html tagů, které jsou rovněž modifikovány rozšiřující komponentou Clarity Design System. V souboru contract.component.ts se vyskytuje zdrojový kód psaný v TypeScriptu, kde je definované třída stránky a metody, které jsou používány. Celá zobrazovaná stránka je rozvržená do čtyř částí, přičemž každá část poskytuje přehledné informace o svém segmentu.

3.7.1 Detail smlouvy

První částí dashboardu jsou detailní informace o smlouvě. Uživatel má možnost, pomocí menu na boku, si zobrazit obecné informace o smlouvě, informace o nastavení rezervované kapacity, kdy pro maloodběr je zobrazováno nastavení jištění a distribuční sazby, kdežto pro velkoodběr je vypsána tabulka s nastavenou měsíční a roční rezervovanou kapacitou pro každý měsíc. Další možností je zobrazení parametrů připojení, zobrazení informací o prodeji energie či informace potřebné k platbě za energii. Dále si uživatel může zobrazit kontakty a faktury, spadající pod danou smlouvu.

Veškeré tyto informace jsou získány při inicializaci stránky, kdy se zavolá metoda GetContract s parametrem ContarctId, který je identifikačním číslem smlouvy a získali jsme ho metodou, která se provedla při kliknutí na kartu smlouvy, kterou jsme chtěli zobrazit. Po zavolání metody GetContract se zavolá rest metoda RestQMethod a provede se spojení s databází pomocí knihovny QEWD. Na straně databáze se zavolá metoda GetContractCP, které se předá parametr s identifikačním číslem smlouvy. Tato metoda nejdřív otevře třídu se smlouvou, která náleží danému identifikačnímu číslu a vrátí konfiguraci cp, která nám vrací veškerá data o smlouvě. Následně se ještě provedou metody GetReservedCapacity, GetEndplaceCP a GetContractEmeterHistory. První z uvedených metod nám vrátí nastavení limitních hodnot rezervované kapacity pro jednotlivé měsíce, další dvě uvedené metody, jsou metody předávající data o odběrném místě a budou probrány v následující kapitole.

```
ClassMethod GetContractCP() As %Library.DynamicObject {
      …
oContract=##class(Ap.NEO.D.Customer.E.Contract).%OpenId(oRequest.id)
      S oResponse.data.Contract=oContract.GetProxyObj(1,"cp")
oResponse.data.Contract.ReservedCapacity=##class(CP.API.Portal).
GetReservedCapacity(oRequest.id)
      S oResponse.data.Endplace = ##class(CP.API.Portal.Endplace).Ge-
tEndplaceCP(oContract)
      S oResponse.data.Invoices=..GetInvoices(oContract)
oResponse.data.Emeters=##class(CP.API.Portal.Endplace).
GetContractEmetersHistory(oRequest.id)
      Q oResponse
}
ClassMethod GetReservedCapacity(ContractId) As %Library.DynamicArray
{
      S list=[]
rs=##class(%Library.ResultSet).%New("CP.API.Portal:ReservedCapacity")
      D rs.Execute(ContractId)
      While rs.Next() {
            S rec={}
oReservedCapacity=
##class(Ap.NEO.D.Customer.E.ContractReservedCapacity).
%OpenId(rs.Get("ID"))
            S YearMonth = oReservedCapacity.Month
            S rec.PerMonth = oReservedCapacity.PerMonth
            …
            S rec.Date = (Month_"."_Year)
            D list.%Push(rec)
      } Q list
```

Zdrojový kód 16 – Metoda pro získání informací o smlouvě zákazníka v Ensemble

Po dokončení výše uvedených metod se odešle odpověď s daty zpět do zákaznického portálu, kde jsou uloženy do proměnné Contract, z které jsou následně vypisovány do dashboardu smlouvy.

3.7.2 Detail odběrného místa

Další část dashboardu obsahuje informace o odběrném místě, které náleží dané smlouvě. Uživatel si zde může zobrazit obecné informace, adresu odběrného místa, seznam elektroměrů, který byly či jsou na odběrném místě nainstalovány a mapu odběrného místa. Tyto informace se rovněž získávají při inicializaci stránky metodou GetContract, jako v přechozí kapitole. Součástí metody GetContractCP, na straně

databáze, je provedení metod GetEndplaceCP a GetContractEmeterHistory, které získávají informace o odběrném místě.

```
ClassMethod GetEndplaceCP(oContract) As %DynamicObject
{
      D ##class(Ap.NEO.D.Endplace.Period.E).GetRelations(
oContract,.oEndplace,.oMeter,.oETrader)
      S tEndplacePx={}
      S tEndplacePx.Active =oEndplace.Active
      S tEndplacePx.Ean =oEndplace.Ean
      S tEndplacePx.Identifier=oEndplace.Identifier
      S tEndplacePx.EndplaceName=oEndplace.Name
      S tEndplacePx.EndplaceId=oEndplace.EndPlaceId
      S tEndplacePx.Note=oEndplace.Note
      S tEndplacePx.Address={}
      S tEndplacePx.Address.Recipient=
oEndplace.Address.Recipient
      S tEndplacePx.Address.Street=oEndplace.Address.Street
      S tEndplacePx.Address.City=oEndplace.Address.City
      S tEndplacePx.Address.ZipCode=oEndplace.Address.ZipCode
      S tEndplacePx.Address.Country=oEndplace.Address.Country
      Q tEndplacePx
}
```

Zdrojový kód 17 - Metoda pro získání informací o odběrném místě v Ensemble

Metoda GetEnplaceCP nejdříve zavolá metodu GetRelations, které předá parametr se smlouvou, podle níž se dohledá odběrné místo, patřící dané smlouvě. Jednotlivé parametry jsou poté předávány proměnné tEndplacePx, která je při návratu naplněna daty o odběrném místě. V metodě GetContractEmeterHistory, získávající data o nainstalovaných elektroměrech, se nejdříve provede sql příkaz, který podle předávaného parametru (id smlouvy) vyhledá identifikační číslo přiděleného odběrného místa a časový úsek, kdy patřilo odběrné místo pod smlouvu. Následně se provede while cyklus, kde se pro každý záznam otevře třída s elektroměry, které se nacházeli či nacházejí na odběrném místě v daný časový úsek, který jsme v prvním kroku zjistili pomocí sql příkazu. Nalezená data o elektroměrech se předají proměnné rec, která je poté vložena do pole list. Pole list se při návratu vrací naplněné daty o elektroměrech patřících odběrnému místu.

```
ClassMethod GetContractEmetersHistory(ContractId) As
%Library.DynamicArray
{
      &sql(SELECT Endplace, TimeBegin, TimeEnd INTO :EndplaceId,:TimeBe-
gin,:TimeEnd FROM Ap_NEO_D_Endplace_Period.E WHERE Contract = :Con-
tractId)
      S rs=##class(%Library.ResultSet).%New("Ap.NEO.D.Endplace.Period.E:
EmetersOnEndplaceT")
      D rs.Execute(EndplaceId, TimeBegin, TimeEnd)
      S list=[]
      While rs.Next() {
            S rec={}
      S oEmeter=
##class(Ap.NEO.D.Device.EMeter).%OpenId(rs.Get("Emeter"))
            S rec.Identifier=oEmeter.Identifier
            S rec.Serial=oEmeter.Serial
            S timeB=rs.Get("TimeBegin")
            S timeE=rs.Get("TimeEnd")
            S rec.timeB=$$$GetTime(timeB,"4.m")
            S rec.timeE=$$$GetTime(timeE,"4.m")
            S rec.timeTest=$$$GetTime("5556729600","4.m")
          S rec.timeTestKonec=$$$GetTime("5559235200","4.m")
            S timeBH=(timeB/86400)
            If $L(timeE) S timeEH=timeE/86400
            Else  S timeEH=$H+30
            D list.%Push(rec)
      }
      Q list
}
```

Zdrojový kód 18 – Metoda získávající záznamy o elektroměrech patřící odběrnému místu v Ensemble

Po dokončení metod, jsou data o odběrném místu a data o nainstalovaných elektroměrech odeslány společně s daty o smlouvě zpět do zákaznického portálu, kde jsou uloženy do proměnné Endplace a pole Emeters.

3.7.3 Naměřené hodnoty

Následující část dashboardu zobrazuje naměřené hodnoty odběru energie v přehledné tabulce. Hodnoty jsou vždy vypisovány pro den nebo rozsah dní, který je vybrán pomocí kalendáře. Při inicializaci stránky je vždy vybrán aktuální den, ale uživatel si může vybrat jakýkoliv předešlý den, až do dne, kdy začala smlouva.

Zároveň si může vybrat rozsah dní, který je omezen na maximálně zobrazitelných sedm dní.

Naměřená data se získávají metodou GetProfile, která je volaná při inicializaci stránky a zároveň také při výběru dne v kalendáři. Po zavolání metody GetProfile se nejprve zkontroluje, jestli se jedná o výpis naměřených hodnot pro jeden den nebo pro více dní. Následně se zavolá rest metoda, které provede spojení s databází pomocí QEWD knihovny a zavolá metodu GetProfileCP, které předá tři parametry. Prvním parametrem je identifikační číslo smlouvy, druhým parametrem je zvolený den, od kterého chceme data vypsat, a v posledním třetím parametru je den, který ukončuje rozsah, do kterého chceme data vyspat. Pakliže chceme vypsat data pouze pro jeden den, tak se po zvolení jednoho dne z kalendáře naplní oba parametry, reprezentující rozsah, stejnou hodnotou a metoda na straně serveru rozezná, že se nejedná o rozsah, ale pouze o výpis naměřených hodnot pro jeden den.

Volaná metoda GetProfileCP získává naměřená data, jak pro vypsání do tabulky, tak zároveň i pro zobrazení v grafu. Při vykonávání metody se v prvním kroku zjistí, z předávaných parametrů, o kterou smlouvu se jedná a rozsah, pro který chceme naměřené hodnoty získat. Následně se vytvoří pole, které obsahuje identifikační čísla elektroměrů, které byly nebo jsou nainstalovány v námi zvoleném rozsahu a z kterých se budou naměřené hodnoty odečítat. Následuje nastavení parametrů, které jsou potřebné pro správné vykreslování grafu, jedná se o nastavení šířky, výšky a definování měřených funkcí, které budou zobrazovány. V dalším kroku následuje for cyklus, který postupuje po patnáctiminutových intervalech, pro které jsou uloženy naměřené hodnoty z elektroměrů. Tyto časový intervaly ukládá do časové osy a naměřené hodnoty ukládá do proměnné gdata, pro které zároveň hledá maximum a minimum, které jsou později využité pro správné vykreslení grafu. Získaná data se v odpovědi vrátí zpět do portálu, kde jsou uloženy do pole gdata, z kterého se následně naměřené hodnoty vypisují do tabulky a zároveň se předávají komponentě, zajišťující vykreslování grafu.

```
ClassMethod GetProfileCP(req = "") As %Library.DynamicArray {
	…
	D ##class(Ap.NEO.D.Endplace.Period.E).GetArrayForContract(
oRequest.ContractId,.arr)
	S k=dataFrom+1 S lk=dataFrom
	For {
		S k=$O(arr(k),-1) Q:k=""
		S $LI(arr(k),3)=lk S $LI(arr(k),4)=k S $LI(arr(k),5)=lk
		S lk=k I k<dataTo S $LI(arr(k),4)=dataTo Q
	}
	…
	For xtime=dataFrom:900:dataTo {
		S time="" If xtime#7200=0 S time=
$P($P(##class(xx.dt.xtime).ToOutput(xtime,8)," ",2),":",1,2)
		If xtime#86400=0 S time=
$P(##class(xx.dt.xtime).ToOutput(xtime,"4.m")," ",1)
		D gdata.timeaxe.%Push(time)
		S xx=$O(arr(xtime+1),-1,rec) S DeviceId=$LG(rec,1)
		S gdata.graph.width = gdata.graph.width + 1
		S rec=..GetProfileRec(DeviceId, xtime)
		D gdata.values.%Push(rec)
		…
	}
```

Zdrojový kód 19 - Část metody pro získání naměřených hodnot z elektroměrů v Ensemble

3.7.4 Graf

Poslední částí dashboardu smlouvy je graf, zobrazující naměřené hodnoty z elektroměru. Nejdříve pro interpretaci grafu byl využita komponenta grafu z PrimeNG, která se pomocí html tagu p-chart vložila do html zdrojového kódu stránky a přiřadil se jí zdroj s daty, kde byly hodnoty pro časovou osu a hodnoty pro vykreslování čar grafu, které reprezentovali naměřené hodnoty funkcí z elektroměru. Toto provedení bylo velice jednoduché a nebylo potřeba nic programovat na straně Angularu, bohužel neslo s sebou dvě zásadní negativa. Prvním negativem byla rychlost vykreslování grafu, která byla značně pomalá a vykreslování naměřených údajů jednoho dne, kde se jedná o devadesát šest čtvrthodinových intervalů, trvalo přibližně osm sekund. Druhým negativem byl vzhled vykresleného grafu, který nelze ve větší míře upravovat a do celkového vzhledu dashboardu nezapadal. Z těchto důvodů bylo usouzeno, že bude vytvořena vlastní komponenta, která bude zprostředkovávat vykreslování grafu.

Zdrojové soubory této komponenty se nacházejí v podadresáři atx-graph v adresáři components. Zde nalezneme tři soubory atx-graph.component.css, atx-graph.component.html a atx-graph.component.ts. První soubor slouží k definování lokálních stylů komponenty, které nejsou v tomto případě použity. V druhém souboru je uložen html zdrojový kód grafu, který obsahuje tagy svg, reprezentující vektorový formát grafu, pomocí něhož je graf vykreslován. V třetím souboru komponenty je zdrojový kód grafu napsaný v TypeScriptu. Nalezneme zde definici selectoru, pomocí něhož je graf vkládán ho html kódu stránky s hlavním dashboardem, dále tu je definovaná třída komponenty a metody, které graf využívá. První takovou metodou je metoda GetValues, která ze získaných naměřených čtvrthodinových hodnot sestavuje svg příkaz, který kreslí čáru grafu. Těchto čar je celkem šest a každá z nich má jinou barvu a reprezentuje jinou měřenou funkci. Modrá barva představuje měřený činný odběr, červená barva odběr ve vysokém tarifu, zelená barva odběr v nízkém tarifu, fialová barva reprezentuje činnou dodávku, oranžová induktivní jalový odběr a žlutá barva kapacitní jalový odběr. Další používanou metodou je ngOnChanges, která mění šířku grafu a časové osy, podle počtu naměřených záznamů. Vytvořením vlastní komponenty, využívající vektorový formát svg pro vykreslování grafu, se docílilo rychlejšího vykreslování, které nyní trvá přibližně jednu sekundu, a hlavně neomezených možností při vytváření vlastního vzhledu grafu.

```
<div #graphContainer
style="position:relative;top:0px;height:330px;width:100%;border:0px solid
Orange;overflow:auto;">
        <svg *ngFor="let rec of data.timeaxe; index as i;" [ngStyle]=
"{stroke:'grey',position:'absolute','width':this.graphWidth+'px','top':
'283px',height:'25px','border':'0px solid Red'}">
            <line [attr.x1]="i*15" y1="10" [attr.x2]="i*15" y2="0" />
            <line *ngIf="rec!='' " [attr.x1]="i*15" y1="25"
[attr.x2]="i*15" y2="-30" style="stroke:#02660a;stroke-width:1" />
            <text *ngIf="rec.length==5" y="25" [attr.x]="(i*15)+3"
style="font-size:12px;stroke:rgb(68, 68, 68);stroke-width:0.5;
font-family:Gruppo;">{{rec}}</text>
            <text *ngIf="rec.length==10" y="25" [attr.x]="(i*15)+3"
style="font-size:12px;stroke:rgb(31, 31, 31);stroke-width:0.5;
font-family:Gruppo;">{{rec}}</text></svg>
```

Zdrojový kód 20 – Část html kódu grafu s použitím vektorového formátu svg v Angularu

4 Závěr

Až na pár výjimek probíhal vývoj aplikace přesně podle plánů a konečná výsledná aplikace je využitelná pro budoucí zákazníky distributorů elektrické energie. Zákazník pomocí aplikace získává přehled o spotřebě elektrické energie, jak v minulých dnech, tak i v čtvrthodinových intervalech aktuálního dne. Odběr elektřiny je zaznamenán v tabulce i v přehledném grafu, u kterého si lze vybrat vždy konkrétní funkce, které se mají zobrazit. Pomocí přehledných informací o odběru, je zákazník schopen reagovat na případné problémy, jako je například překročení limitních hodnot. Další možností zákaznického portálu je například upravování smlouvy podle potřeb zákazníka či zobrazení veškerých dostupných údajů a faktur, které jsou stažitelné přímo z portálu.

Spojením javascriptového frameworku Angular a objektově orientované databáze InterSystem Caché, se docílilo příznivé rychlosti při přenosu velkoobjemových dat, která byla pro tuto problematiku stěžejní. Vyřešením prvního problému, kdy do přímé komunikace mezi aplikací a databází vstoupil další prvek, QEWD knihovna, se dosáhlo nezávislosti na databázové platformě a možnosti použití služeb Active Directory a FileStream, které by se jinak museli na straně databáze doprogramovat. Další problém nastal ke konci vývoje, kde se řešilo zobrazování naměřených hodnot v grafu. Prvním pohodlným řešením, bylo použití již před vytvořené komponenty grafu z PrimeNG, kdy stačilo pouze grafu předat data a komponenta se sama postarala o vykreslení grafu. Toto řešení bylo velmi jednoduché, bohužel se ukázalo, že vykreslovací doba grafu je příliš dlouhá. Problém byl vyřešen naprogramováním vlastní komponenty, která vykresluje graf pomocí svg vektorového formátu a vykreslovací doba byla přibližně zkrácena na desetinu původní hodnoty. Již při vývoji jednotlivých částí aplikace, byly veškeré funkce aplikace testovány na databázi s daty, aby se předešlo pozdějšího hledání problému a konečná verze zákaznického portálu byla řádně odzkoušena na ostrých datech, kde nedošlo k žádnému problému.

Jelikož se jedná o zákaznický portál, který má poskytovat zákazníkovi informace spojené s odběrem elektrické energie, lze samotnou aplikaci jednoduše rozšířit o informace týkající se odběru plynu či tepelné energie. Z informačního pohledu, kde se jedná o zobrazování údajů zákazníkovi, které jsou vyžadované vyhláškou, či zobrazování naměřených hodnot z měřáků, se rozšíření o plyn či tepelnou energii tolik neliší od poskytování údajů o elektrické energie. Z tohoto důvodu je možné zákaznický portál rozšířit pouze o výběr zobrazované oblasti a ostatní komponenty lze, bez větších úprav, použít i pro rozšíření o plyna tepelnou energii. Tímto rozšířením by se zákaznický portál stal komplexním řešením pro různé dodavatele energií, kteří chtějí poskytovat svým zákazníkům elektronickou možnost podávání a stahování smluv či faktur a zároveň poskytovat přehled o jejich odběru energie.

Použitá literatura

[1] HASSMAN, Martin. *Proč používat javascriptový framework?* [online]. 30.9.2008 [cit. 2018-04-17]. Dostupné z: https://www.zdrojak.cz/clanky/proc-javascriptovy-framework/

[2] Developer Survey Results 2016. *Stack Overflow* [online]. [cit. 2018-04-17]. Dostupné z: https://insights.stackoverflow.com/survey/2016

[3] Developer Survey Results 2017. *Stack Overflow* [online]. [cit. 2018-04-17]. Dostupné z: https://insights.stackoverflow.com/survey/2017

[4] VÁŠKO, Ondřej. *Top 10 frameworků pro moderní frontend* [online]. 10.2.2017 [cit. 2018-04-17]. Dostupné z: https://www.wdt.cz/inovace/top-10-frameworku-pro-moderni-frontend-a589ddee72867455fd9c17a2d

[5] *Angular: One framework. Mobile & desktop.* [online]. [cit. 2018-04-17]. Dostupné z: https://angular.io/

[6] *React: A JavaScript library for building user interfaces* [online]. [cit. 2018-04-17]. Dostupné z: https://reactjs.org/

[7] *Ember: A framework for creating ambitious web applications.* [online]. [cit. 2018-04-17]. Dostupné z: https://www.emberjs.com/

[8] *Meteor: THE FASTEST WAY TO BUILD JAVASCRIPT APPS.* [online]. [cit. 2018-04-17]. Dostupné z: https://www.meteor.com/

[9] Developer Guide: Data Binding. *AngularJS* [online]. [cit. 2018-04-17]. Dostupné z: https://docs.angularjs.org/guide/databinding

[10] PANDADIS, Jakub. *Analýza potřeb zákaznického portálu pro odběratele elektrické energie*. Liberec, 2017. Bakalářský projekt. Technická univerzita v Liberci. Vedoucí práce Ing. Jan Kraus, Ph.D.

[11] Caché: You'll make breakthroughs in data management and rapid development. *InterSystems* [online]. [cit. 2018-04-17]. Dostupné z: http://www.intersystems.com/cz/our-products/cache/cache-overview/

[12] Caché: The Database for Applications That Matter. *InterSystems* [online]. [cit. 2018-04-17]. Dostupné z: https://www.intersystems.com/products/cache/#technology

[13] *Node.js* [online]. [cit. 2018-04-17]. Dostupné z: https://nodejs.org/

[14] *Visual Studio Code: Code editing. Redefined.* [online]. [cit. 2018-04-17]. Dostupné z: https://code.visualstudio.com/

[15] Ensemble: You'll make breakthroughs in rapid connectivity. *InterSystems* [online]. [cit. 2018-04-17]. Dostupné z: http://www.intersystems.com/cz/our-products/ensemble/ensemble-overview/

[16] QuickStart. *Angular* [online]. [cit. 2018-04-17]. Dostupné z: https://angular.io/guide/quickstart

[17] *Clarity Design System* [online]. [cit. 2018-04-17]. Dostupné z: https://vmware.github.io/clarity/

[18] *PrimeNG: The Most Complete User Interface Suite for Angular* [online]. [cit. 2018-04-17]. Dostupné z: https://www.primefaces.org/primeng/

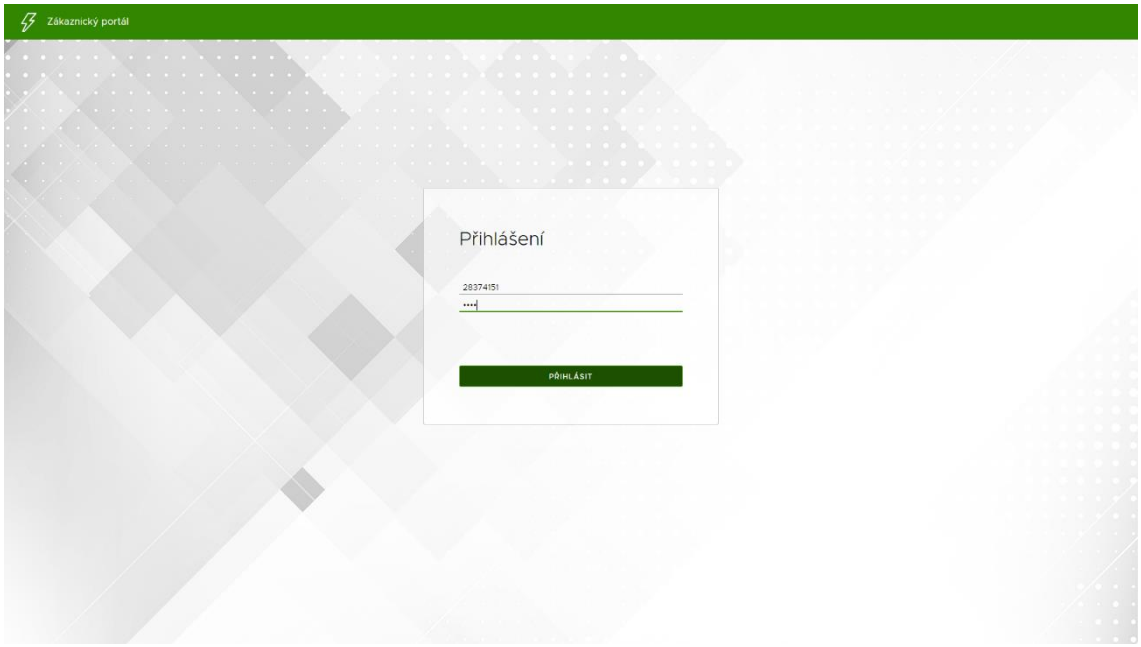
[19] *Font Awesome: Icons. Easy. Done.* [online]. [cit. 2018-04-17]. Dostupné z: https://fontawesome.com/

[20] MALÝ, Martin. *REST: architektura pro webové API* [online]. 3.8.2009 [cit. 2018-04-21]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

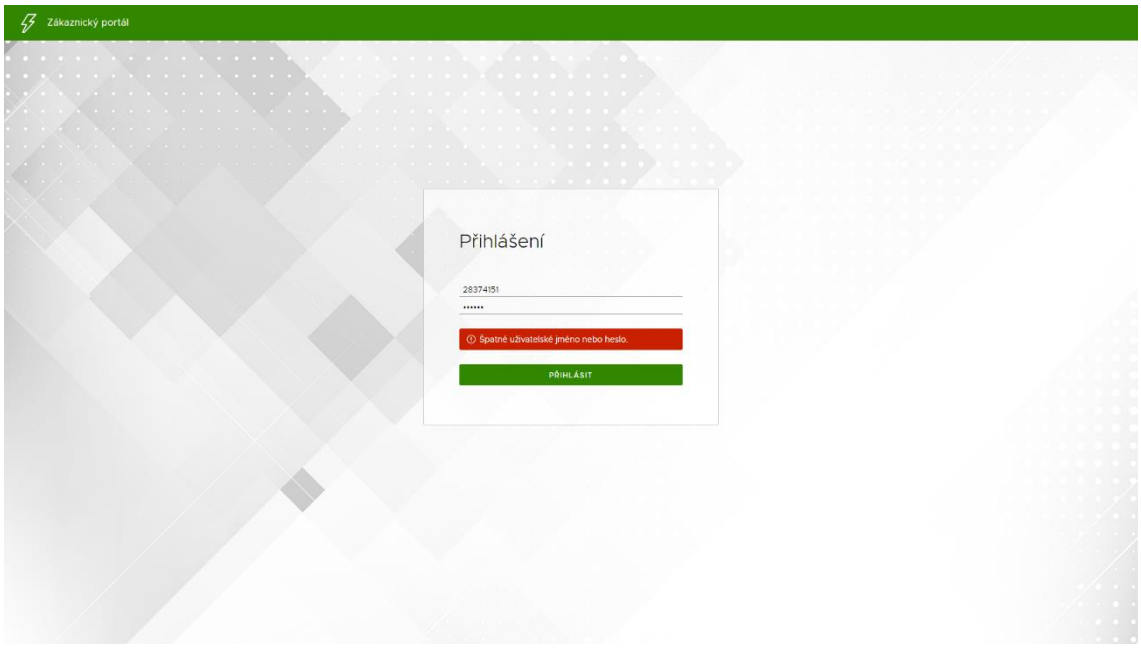
[21] HANÁK, Drahomír. *Stopařův průvodce REST API* [online]. [cit. 2018-04-21]. Dostupné z: https://www.itnetwork.cz/nezarazene/stoparuv-pruvodce-rest-api

[22] *QEWDJS: Quality Enterprise Web Development* [online]. [cit. 2018-04-21]. Dostupné z: http://qewdjs.com/

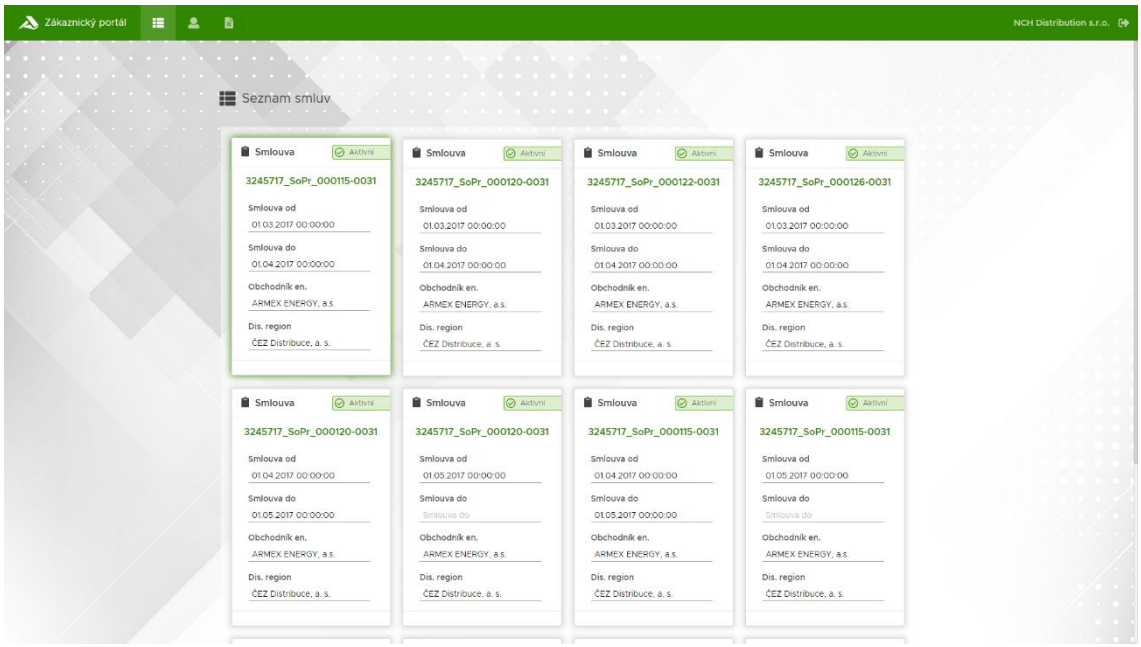
A Obrázky aplikace zákaznického portálu



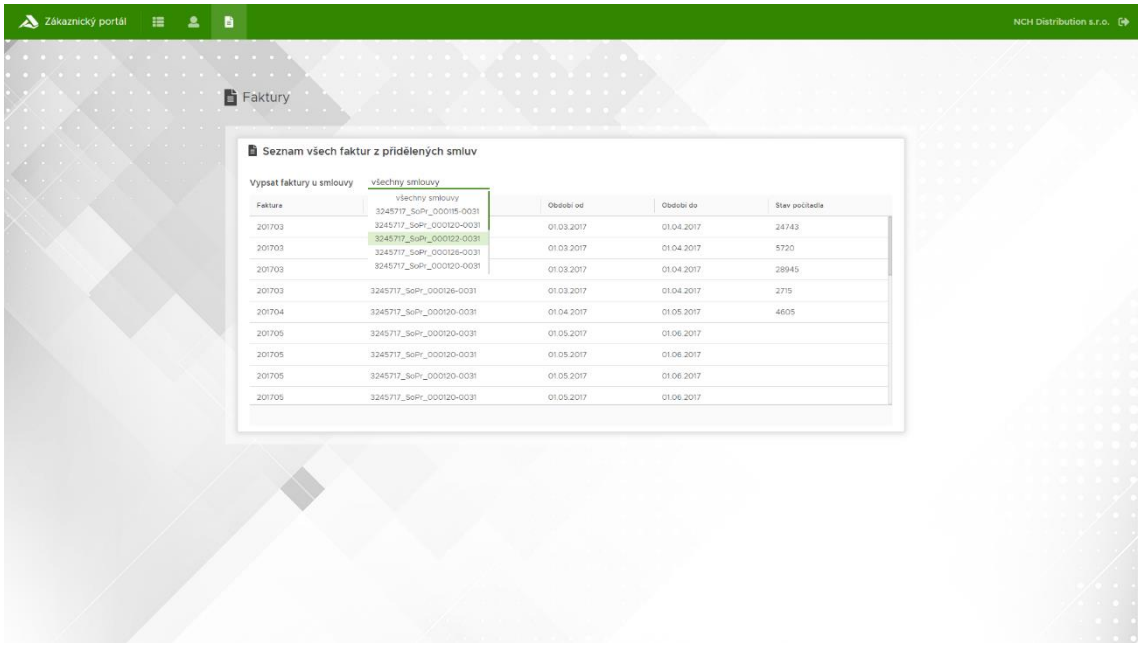
A Obrázek 1 - Přihlašovací stránka do zákaznického portálu



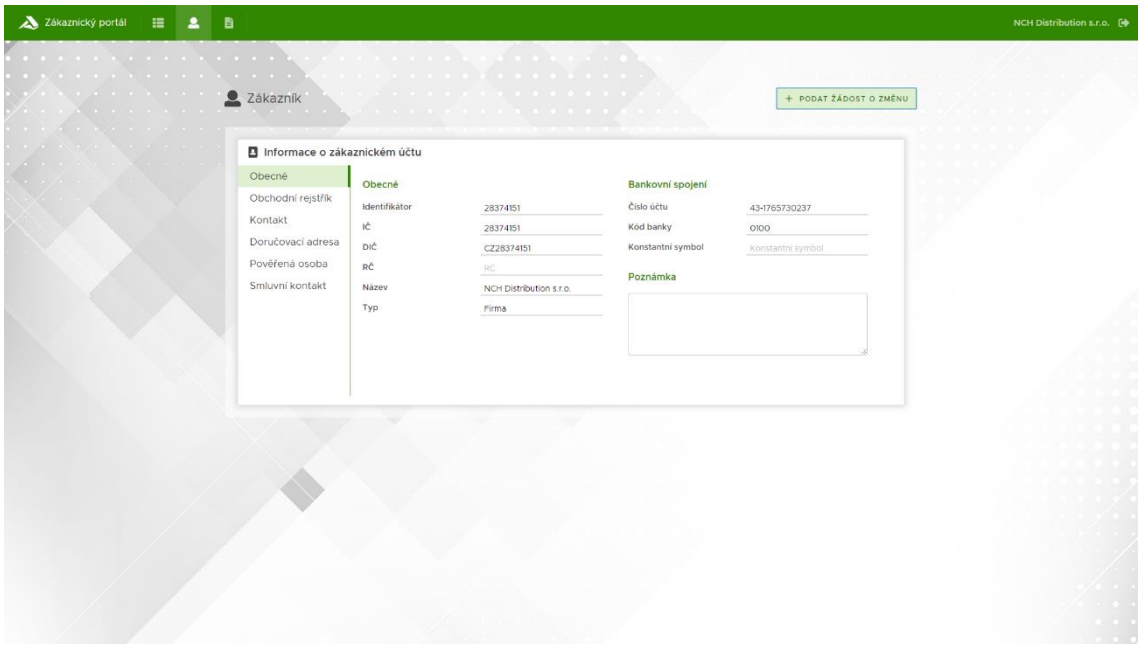
A Obrázek 2 - Upozornění na špatně zadané heslo nebo uživatele při přihlášení



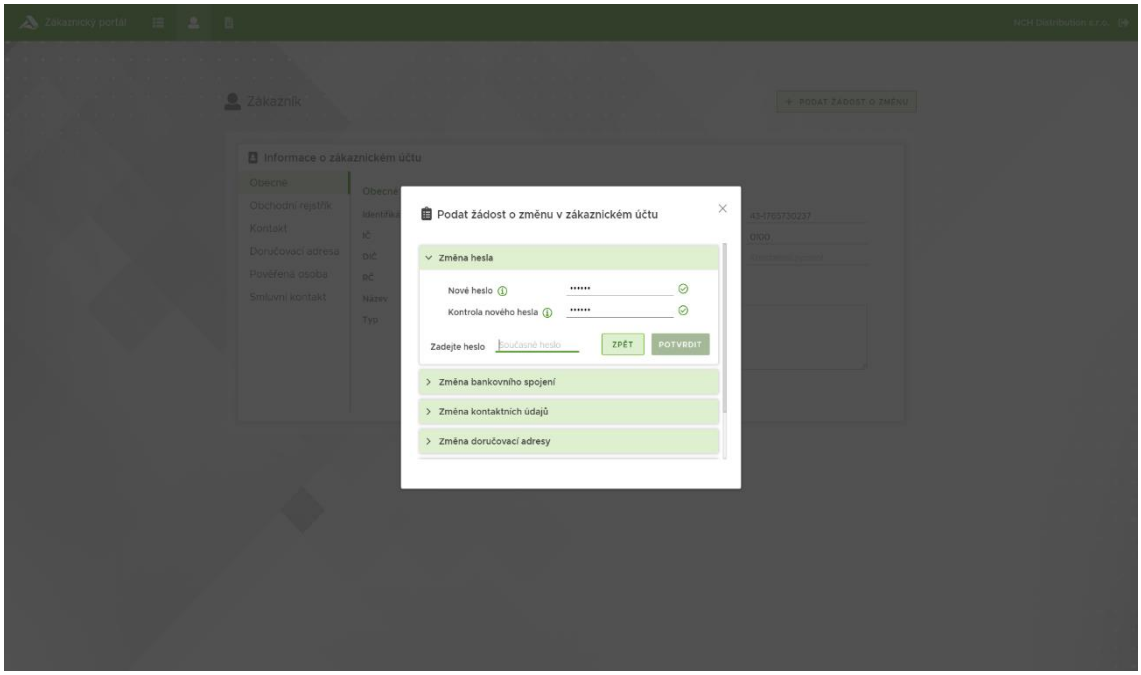
A Obrázek 3 - Stránka s výpisem všech smluv zákazníka



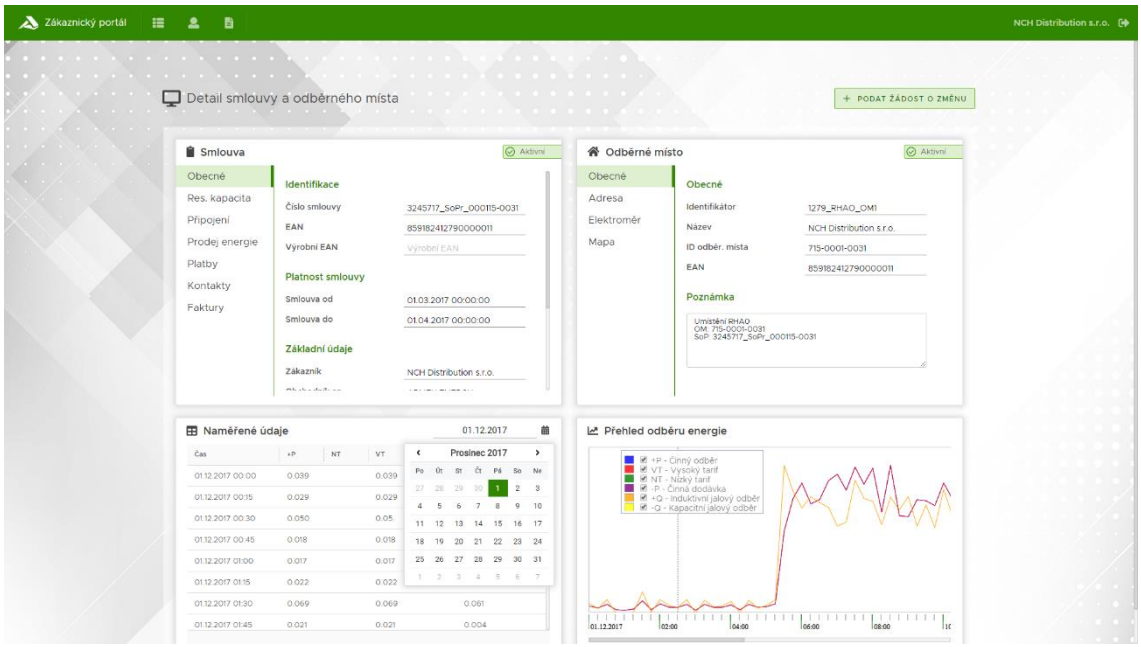
A Obrázek 4 - Stránka s výpisem všech faktur



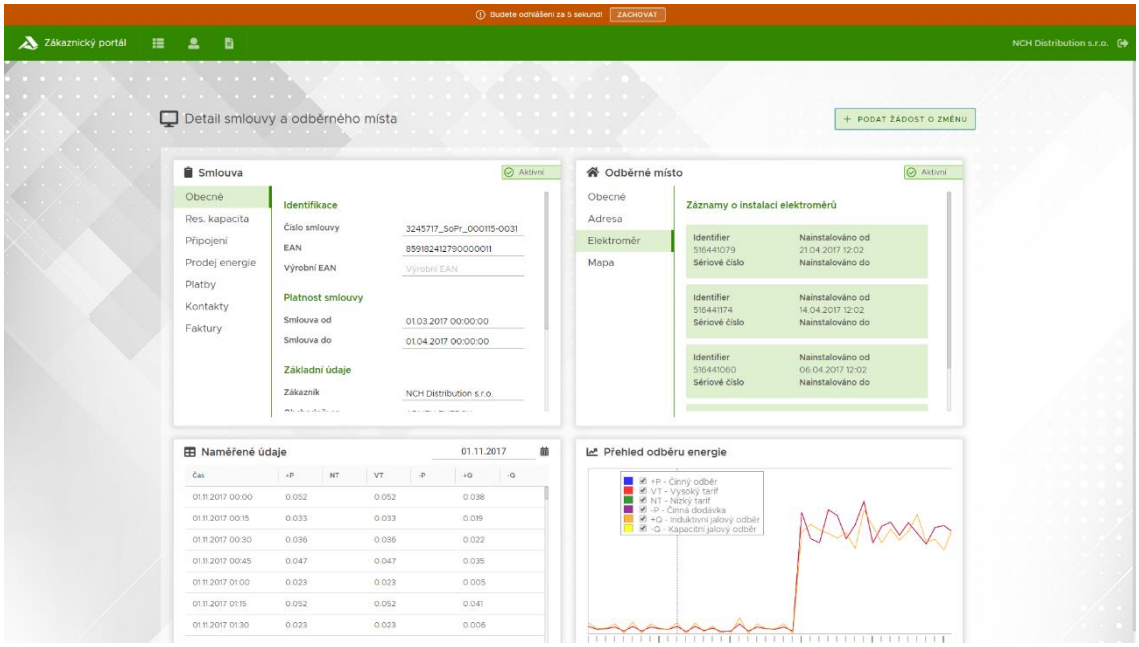
A Obrázek 5 - Stránka s informacemi o zákazníkovi



A Obrázek 6 - Otevřené modální okno pro podání žádosti na změnu údajů o zákazníkovi



A Obrázek 7 - Stránka s detailem smlouvy a odběrného místa



A Obrázek 8 - Upozornění uživatele na následné odhlášení při neaktivitě

B Obsah přiloženého CD

- text bakalářské práce
  - bakalarska_prace_2018_Jakub_Pandadis.pdf
  - bakalarska_prace_2018_Jakub_Pandadis.doc
  - kopie_zadani_bakalarska_prace_2018_Jakub_Pandadis.pdf
- zdrojový kód programu
  - soubory aplikace napsané v Angularu