Střední průmyslová škola na Proseku
190 00 Praha 9, Novoborská 2

OBOR

18-20-M/01 INFORMAČNÍ TECHNOLOGIE
ŠVP: Správa sítí a IT bezpečnost

MATURITNÍ PROJEKT

TÉMA PRÁCE

Aplikace pro zaznamenávání klientů a jejich internetových služeb

2023/2024 Pokorný Vojtěch

PROHLÁŠENÍ

Prohlašuji, že jsem svou maturitní práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 1. 3. 2024 ……

podpis

Střední průmyslová škola na Proseku
190 00 Praha 9, Novoborská 2

ZADÁNÍ PRAKTICKÉ MATURITNÍ PRÁCE

Žák:	Vojtěch Pokorný
Obor:	18-20-M/01 – Informační technologie
Školní rok:	2023/2024
Téma práce:	Vývoj aplikace pro zobrazování dat
Název práce:	Aplikace pro zaznamenávání klientů a jejich internetových služeb
Ved. práce:	Ing. Jiří Šilhán
Oponent:	bude jmenován ředitelem školy v souladu s vyhláškou 177/2009 Sb.

Termín odevzdání – řádný termín:	**1. 3. 2024 do 12:00 hodin**
Termíny odevzdání – opravné termíny:	podzimní opravný termín: nejpozději poslední pracovní den v červnu do 12:00 hodin jarní opravný termín: nejpozději první pracovní den v březnu do 12:00 hodin
Délka obhajoby maturitní práce před maturitní komisí:	15 minut včetně doplňujících otázek

Poučení: *Dle vyhlášky 177/2009 Sb. § 15 odst. 7 – Neodevzdá-li žák pro vážné důvody práci v termínu stanoveném podle odstavce 1 písm. b), omluví se písemně řediteli školy nejpozději v den stanovený pro odevzdání maturitní práce; uzná-li ředitel školy omluvu žáka, určí žákovi náhradní termín pro odevzdání maturitní práce. Pokud žák maturitní práci neodevzdá v termínu podle odstavce 1 písm. b) bez písemné omluvy s uvedením vážných důvodů nebo pokud mu omluva nebyla uznána, posuzuje se, jako by danou zkoušku vykonal neúspěšně.*

Dle školského zákona č. 561/2004 Sb. § 79 odst. 7 - Profilová část maturitní zkoušky je veřejná s výjimkou zkoušek konaných formou písemné zkoušky a jednání zkušební maturitní komise o hodnocení žáka; zkoušky konané formou praktické zkoušky jsou neveřejné v případech, kdy je to nutné z důvodu ochrany zdraví, bezpečnosti práce a u zdravotnických oborů také z důvodu ochrany soukromí pacienta.

Pokyny pro vypracování:

- vypracování rešerší pro seznámení s řešenou tématikou V práci musí být vypracovány rešerše v takovém rozsahu, aby byla odůvodněna každá část návrhu od volby koncepce řešení až po volbu jednotlivých komponent, či návrh programu. Zároveň práce nesmí obsahovat rešerše nadbytečné, které s návrhem nesouvisí. Minimální počet rešerší jsou dvě.
- vytvoření technické dokumentace (technické řešení, požadavky, postup instalace, …)
- vytvoření uživatelské dokumentace (ukázka použití, manuály, …)
- plakát ve formátu A1 prezentující práci
- prezentace pro obhajobu práce

Požadavky:

- vypracování jednoduchého průzkumu trhu a stanovení základní požadavků (funkce, využití, …), pokud je to možné a daná práce toto umožňuje
- webová aplikace pro ISP - zaznamenávání klientů a jejich služeb
- automatická komunikace systému s klientem
- ukázka nasazení aplikace v testovacím prostředí

Střední průmyslová škola na Proseku
190 00 Praha 9, Novoborská 2

Hodnocení:

- výsledná známka z maturitního projektu s obhajobou se skládá z hodnocení:
  - hodnocení v závěrečném posudku vedoucího maturitního projektu
  - hodnocení v závěrečném posudku oponenta maturitního projektu
  - hodnocení obhajoby maturitního projektu před maturitní komisí

Hodnocení práce – plagiátorství:
Odevzdané textové části práce budou posouzeny systémem na kontrolu plagiátů odevzdej.cz. V případě míry shody přesahující **15 %** bude práce posouzena předmětovou komisí a výsledek posouzení bude poté předán k rozhodnutí maturitní komisi. Pokud se ukáže při hodnocení práce, či při samotné obhajobě, že je práce plagiátem, maturitní komise rozhodne, že práce bude hodnocena známkou nedostatečný.

Kritéria hodnocení maturitního projektu:

- samostatný a tvůrčí přístup k práci
- dodržování stanovených termínů
- prezentace dosažených výsledků projektu při konzultacích
- dodržení stanoveného rozsahu práce – minimálně pět citovaných zdrojů (nelze citovat web Wikipedia), alespoň dvě témata pro rešerši
- kvalita vypracovaných rešerší
- dodržení typografických pravidel
- kvalita provedení praktické části práce
- splnění pokynů k vypracování
- prezentace výsledků projektu (plakát) a schopnost obhajoby práce (prezentace)

Hodnotící kritéria praktické části maturitního projektu:

- analýza cílového uživatele / prostředí / UX a návrh architektury aplikace - normalizace a uml (databáze, OOP) (15)
- dokumentace, popis API, uživatelská dokumentace, popis instalace a nasazení a popis ukázek (15)
- kvalita kódu a správné použití nástrojů (25)
- připravenost aplikace pro nasazení v cílovém prostředí (15)

Rozsah práce:

- minimální rozsah textové části práce (rešerše a popis praktického řešení) je 15 normostran textu (bez formálních částí – obsah, literatura atd.)
- minimální rozsah praktické části je stanoven pokyny k vypracování práce, tj. splněním cílů práce
- V případě zavedení distanční výuky, nebo nařízení karantény v předmětu Projekt, trvající déle než 45 kalendářních dnů (vč. období prázdnin), nemusí žák odevzdávat fyzický prototyp, který je součástí praktické části práce. V tomto případě bude při hodnocení kladen důraz na dokumentační část projektu (výkresy, modely, schémata, simulace, ...), podle které musí být prototyp realizovatelný.

Počet vyhotovení práce:

- maturitní práce bude odevzdána v elektronické podobě
- veškeré uložené textové dokumenty, včetně příloh (i fotodokumentace a videozáznamů), budou odevzdány v původním formátu (zdrojový formát např. .DOCX) i ve formátu .PDF.
- textová část dokumentu bude obsažena v jednom souboru a bude obsahovat všechny formální části (titulní strana, anotace, obsah atd.), pro zadání bude v dokumentu jedna nečíslovaná stránka
- elektronická verze práce (včetně prezentace) bude uložena na Google Classroom (přesné pokyny k odevzdání budou zaslány žákům na školní email, nebo prostřednictvím IS v průběhu února 2024).

V Praze dne 25. 10. 2023

.....
Ing. Lukáš Procházka
ředitel školy

převzal dne: 25.10.2023

podpis žáka: [signature]

PODĚKOVÁNÍ

Chtěl bych poděkovat Ing. Jiřímu Šilhánovi za rady a konzultace týkající se projektu. Dále bych chtěl poděkovat firmě Prestonet za rady a poskytnutí infrastruktury pro nasazení aplikace a databáze vytvořené v rámci projektu.

ANOTACE

Jméno autora	Vojtěch Pokorný
Název maturitní práce	Aplikace pro zaznamenávání klientů a jejich internetových služeb
Rozsah práce	103 s. + 3 s. příloh
Školní rok vyhotovení	2023/2024
Škola	Střední průmyslová škola na Proseku
Vedoucí práce	Ing. Jiří Šilhán
Využití práce	Zaznamenávání klientů a jejich internetových služeb v prostředí konkrétního poskytovatele internetových služeb.
Abstrakt	Maturitní práce je zaměřena na vývoj interní webové aplikace, která slouží pro zaznamenávání klientů a jejich internetových služeb. Zaměřuje se na konkrétní řešení a požadavky poskytovatele internetu, u kterého bude aplikace nasazena.
Klíčová slova	webové stránky, ISP, internetová služba, relační databáze

ANNOTATION

Autor	Vojtěch Pokorný
Title of graduation work	Application for recording clients and their internet services
Extend	103 p. + 3 p. of appendices
Academic year	2023/2024
School	Střední průmyslová škola na Proseku
Supervisor	Ing. Jiří Šilhán
Application	Recording clients and their internet services in the environment of a specific internet service provider.
Abstrakt	The graduation thesis focuses on the development of an internal web application that is used to record clients and their internet services. It focuses on a specific solution and the requirements of the internet provider where the application will be deployed.
Key words	websites, ISP, internet service, relational database

Obsah

1 Úvod

Každý poskytovatel internetu potřebuje zaznamenávat informace o svých internetových službách, ať už technických nebo těch finančních. Dnes je velmi běžné pro tyto účely využít aplikaci, protože se tím zajistí vyšší dostupnost dat, vyšší integrita dat a ulehčí se tím mnoho interních procesů ve firemním prostředí.

Cílem projektu je vytvořit jednoduchou webovou aplikaci pro zaznamenávání výhradně technických informací o internetových službách a klientech, kterým jsou tyto služby poskytovány. Aplikace má tedy sloužit výhradně jako nástroj pro techniky pracující u poskytovatele.

Projekt je navrhován a tvořen pro poskytovatele internetu Prestonet, který sídlí v samotném centru Prahy. Všechny funkce a postupy budou tedy navrženy dle požadavků, které si tento poskytovatel určí a budou s ním v průběhu vývoje konzultovány. Aplikace vytvořená v rámci projektu se po odevzdání bude také nadále vyvíjet a posouvat, očekává se tedy i budoucí spolupráce s autorem.

V projektu budou nejdříve popsána odborná témata týkající se samotného projektu. Těmito tématy budou výhradně systémy a aplikace pro poskytovatele internetu, relační databáze a webové technologie. V nich budou představeny témata, která jsou důležitá pro praktickou část a její pochopení. Dále se v projektu bude nacházet uživatelská a technická dokumentace, kde jsou popsány konkrétní záležitosti návrhu, struktury, nasazení a použití aplikace.

2 Systémy a aplikace pro ISP

ISP (Internet Service Provider – poskytovatelé internetu) používají různé počítačové systémy a aplikace. Ať už specificky určené pro poskytovatele internetu a kvalitní správu jejich služeb nebo obecnější systémy, které může využít jakákoliv společnost, a to například aplikace pro správu zdrojů a financí.

2.1 ISP – Internet Service Provider

Poskytovatel internetu je společnost, která nabízí a dodává jednotlivcům i jiným společnostem přístup k internetu a popřípadě i jiné související služby. Mezi tyto související služby patří například hostování datových a webových serverů, IT podpora nebo jiný outsourcing služeb a infrastruktury.

Poskytovatelé musí mít vybavení a infrastrukturu, aby mohli poskytovat kvalitní internetové služby. Tato infrastruktura bývá velmi komplexní a složená z mnoha různých zařízení. To, jaká přesně zařízení má poskytovatel v konkrétní infrastruktuře, záleží až na jeho uvážení. To znamená, jaké si vybere výrobce těchto zařízení, typy zařízení, zda bude pokládat optické kabely, nebo si je bude pronajímat a tak dále. Jak se infrastruktura postupně upravuje a vyvíjí je poměrně složitý proces a čím je infrastruktura větší a komplexnější, tím je složitější a dražší ji změnit, či upravit.

Dále jsou poskytovatelé děleni na pomyslné úrovně, dle jejich velikosti, rozsahu a úrovně služeb. První a nejvyšší úrovní jsou převážně společnosti s mezinárodním rozsahem, které vlastní většinu infrastruktury, kterou používají. Mezi Společnosti, které se dají do této úrovně zařadit patří například O2 nebo Vodafone. Do druhé úrovně zapadají regionální poskytovatelé, kteří výhradně propojují poskytovatele první a třetí úrovně, a také poskytují služby spotřebitelům a komerčním zákazníkům. Poslední, tedy pomyslná třetí úroveň poskytovatelů využívá převážně cizí infrastrukturu a zaměřují se na domácnosti a menší obchody. Tyto úrovně jsou čistě orientační tudíž jsou poskytovatelé, kteří přímo nezapadají do určité úrovně, ale jsou například na hraně.

Aby měl sám poskytovatel přístup k internetu a mohl ho poskytovat svým zákazníkům musí být sám někde připojen k internetu. Nejčastějším způsobem bývá peering, což je dohoda mezi poskytovateli, díky které si navzájem umožňují směrovat provoz skrze svoje sítě. Další možností je poté využít poskytovatele třetích stran, aby je navzájem připojil. Tuto službu často

poskytují datová centra, kde mají své zařízení stovky poskytovatelů. Datové centrum bývá jedno zabezpečené místo určené pro uložení fyzické infrastruktury firem, propojení této infrastruktury a poskytování dalších služeb týkajících se fyzické IT infrastruktury. Firmy mívají pronajaté jednotky či desítky rackových skříní pro uložení jejich zařízení. Některé velké firmy, pak už mívají pronajaté celé místnosti nebo mají dokonce vlastní datová centra. Jedno z největších datových center v ČR je Ce Colo, které se nachází na Praze 10. (1) (2)

2.1.1 Způsoby poskytování internetu

Internet je poskytován různými způsoby přenosu dat. To, jakým způsobem se můžete k internetu připojit přímo vy, záleží převážně na lokalitě vašeho bydliště. Záleží především na tom, jací poskytovatelé v dané lokalitě internet poskytují a jakou mají v daném místě infrastrukturu. Mezi nejčastější typy připojení pak patří následující. (1)

Optické vlákno

Připojení skrze optické vlákno je v dnešní době nejrychlejší a nejlepší řešení. Nejmodernější vlákna totiž mohou dosahovat rychlostí přenosu dat až v desítkách Gb/s. Optické sítě se ale ve většině České republiky teprve budují, což je asi největší omezení této technologie. U optiky se vyvinuly další na ní postavené technologie jako např. PON (Passive optical network), které umožňují poskytovatelům budovat optické sítě rychleji a s menšími náklady na výstavbu sítě. (1)

DSL – Digital Subscriber Line Transceiver

Ve zkratce se jedná o připojení klientů k internetu pomocí telefonních kabelů. DSL je velmi zastaralá a nespolehlivá technologie, která převážně využívá staré telefonní infrastruktury. Telefonní kabely, které se používají bývají staré desítky let a nebyly pro poskytování internetu uzpůsobeny. Připojení je zpravidla velmi pomalé, a to okolo 20 Mb/s, i přesto je v dnešní době tento typ připojení velmi rozšířený, díky své malé ceně a už zhotovené infrastruktuře. Nové sítě pro tuto technologii už ale nikdo nebuduje a je jen otázkou času, kdy tato technologie upadne v zapomnění. (1)

4G, LTE a 5G

Toto připojení je nejčastější u mobilních zařízení, protože je postavené na mobilních sítích. V dnešní době se tyto technologie začaly používat také pro připojení celých klientských domácností či menších firem. To vše díky tomu, že pokrytí mobilních sítí je v dnešní době

velmi dobré a připojení je pro poskytovatele ve větším měřítku levnější. Rychlost těchto sítí, ale nebývá příliš velká. (3)

2.1.2 Strategie budování sítě a služeb

Poskytovatelé také mají různé strategie a přístupy pro budování jejich sítě. Neustále se vyvíjejí, řeší nové problémy a soutěží mezi sebou o zákazníky. Každý poskytovatel zvolí trochu jinou strategii, která je zrovna pro něj ta nejefektivnější. Jednotliví poskytovatelé se většinou zaměřují na jeden nebo více typů připojení klientů. Jak přesně to má daný poskytovatel nastavené, a jak rozvíjí svou síť, záleží na mnoha faktorech. Na tom, kde poskytuje internet, komu poskytuje internet, za jakou cenu ho poskytuje a tak dále. Poskytovatel také musí často předpovídat a spočítat si, zda se mu dané budování infrastruktury vyplácí nebo alespoň do budoucna vyplácet bude.

Nejčastěji se v dnešní době budují bezdrátové sítě, a to hlavně kvůli cenové stránce věci. To neplatí pouze pro odlehlé a menší oblasti, ale také pro města. Pronájem nebo výstavba optické či jiné kabelové sítě je v dnešní době totiž velice obtížná a drahá. Takže se všude využívají point-to-point jednosměrné antény, dnes nejčastěji 60 GHz, pro přípojky baráků, a dále také mobilní sítě a technologie jako 4G a 5G. Hlavní nevýhodou bezdrátových sítí je jejich nespolehlivost díky útlumu na trasách. To znamená, že například povětrnostní podmínky a počasí mohou hodně ovlivnit stabilitu a rychlost sítě, takže poskytovatel klientům nemůže poskytnout vysoké SLA (Service Level Agreement), což smluvně udává, kolik procent času v měsíci služba musí běžet a být funkční. Proto je lepší u klientů, kteří potřebují, či chtějí, aby internet fungoval non-stop využít jiného řešení než bezdrátu, ideálně optické sítě.

Budování optické sítě je poměrně náročný proces, a to, zda se v dané oblasti budování vyplatí, je také obtížné rozhodování. Jsou tu sice možnosti pronájmu infrastruktury od jiných poskytovatelů či společností, ale to se ve většině případů vůbec nevyplácí, pokud s danou společností poskytovatel dlouhodobě nespolupracuje. Budování vlastní optické sítě sice vyžaduje více zdrojů, ale jedná se o velkou investici do budoucna, kdy pak poskytovatel sám může infrastrukturu pronajímat, a také může klientům poskytovat stabilnější a rychlejší internet. Tím pádem samozřejmě i dražší.

Připojení přes optickou síť se většinou vyplatí pouze v hustěji zabydlených oblastech, kde se nachází více lidí, firmy a zároveň bohatší klienti, kteří jsou ochotni si připlatit za dražší

internet. Bezdrátové připojení potom spíše u domácností a menších klientů, kde se ceny pohybují v menších částkách a pro jejich velké množství se poskytovateli nevyplatí pro všechny budovat kabelovou infrastrukturu.

Pro klienty je důležité, aby jejich poskytovatel naslouchal jejich přáním a problémům. Mít kvalitní podporu pro své klienty je jeden z rozhodujících faktorů, zda u vás klient zůstane nebo přejde ke konkurenci. Ať už formou helpdesku nebo pouhého telefonního čísla, vždy musí být pro klienta vytvořeno prostředí pro něj co nejvíce výhodné, kde může podat zpětnou vazbu. Pokud je poskytovatel větší je zvykem, že má zaměstnance jejichž úkolem je pouze odpovídat a pomáhat klientům. Pokud daný poskytovatel má dost klientů a zdrojů do přívětivého prostředí určeného pro klienty se vyplatí investovat. Dále také může poskytovatel investovat do pokročilejších řešení jako je například aplikace pro klienta, kde si může poupravit své služby sám, bez pomoci samotného poskytovatele.

Dalším tahákem pro klienty můžou být bonusové služby k samotnému internetu. Patří mezi ně například správa a monitoring jejich sítě, serverový hosting nebo non-stop podpora. Zde už záleží pouze na poskytovateli, zda se mu vyplatí nakoupit potřebný hardware a zaměstnávat někoho, kdo danou věc umí nastavit a spravovat. Jinou možností je také přeprodej těchto služeb, kdy může poskytovatel službu pouze přeprodávat od někoho jiného například internetovou televizi. V tom případě ho to skoro nic nestojí, ale zase nevydělá tolik peněz. (4)

2.1.3 Využití systémů a aplikací

V dnešní době už si žádná společnost nevystačí s tunou papírů, složek a desek, výjimkou není ani poskytovatel internetu. Aby mohl poskytovatel svoji infrastrukturu, co nejlépe a nejefektivněji spravovat a měl vyřešené i všechny svoje účetní a jiné interní procesy, musí využít pomoci systémů a aplikací, které byli právě pro z těchto důvodů vyvinuty. Neexistuje žádný poskytovatel, který by nějaký takový systém nebo aplikaci nevyužíval. Na tom, které přesně už záleží pouze na daném ISP.

2.2 Problematika výběru aplikací a systémů

Každá IT firma, poskytovatel či spolek využívají různou kombinaci systému a aplikací, aby zajistili správný interní běh jejich společnosti. V dnešní době existuje mnoho možností, z kterých mohou společnosti vybírat, proto je velmi důležité, aby bylo při výběru vymezeno,

přesně to, co daná společnost hledá a dle toho vybrat správnou kombinaci zrovna pro konkrétní požadavky a řešení dané společnosti. A právě tento počáteční výběr může být kamenem úrazu pro úspěch, pokud je ze začátku vybráno nevhodné řešení.

Největším problémem, poté není výběr jiné vhodnější aplikace či systému, ale migrace informací a procesů z jednoho systému do druhého. Náročnost a problematika této migrace závisí na tom, kolik interních procesů je závislých na daném systému a zda jsou v daném systému ukládaná nějaká důležitá data, která by se musela migrovat. To, zda by se musela migrovat samozřejmě záleží také na tom, jestli jsou daná data uložena pouze v daném systému.

Existuje mnoho řešení, ať už jedno komplexní či více jednodušších, kdy se různé problematiky řeší odděleně. Vždy se ale dané aplikace či systémy zaměřují více na určitou problematiku. Tím je myšleno, že existují sice desítky systémů pro poskytovatele, ale každý má lepší řešení v něčem jiném a zaměřuje se primárně na něco jiného. Také vždy není nejlepším řešením použití pouze jedné aplikace či systému z důvodu, že je poté poskytovatel kriticky vázaný na daného zprostředkovatele. I zde je totiž potřeba diverzifikovat.

Výběr tohoto softwaru jde ruku v ruce také s výběrem fyzického hardwaru. Dle výběru různých výrobců síťového, serverového a stolního hardware se vymezuje využití různých systémů, které daní výrobci nabízí nebo alespoň podporují jejich běh a funkcionalitu přímo na jejich nebo nad jejich hardwarem.

V dnešní době také začíná být u firem velmi populární vývoj vlastního softwarového řešení, a to ať už z finančních nebo procesních důvodů. Většinou se ale k tomuto řešení společnost uchýlí ve chvíli, kdy potřebuje v aplikaci konkrétní řešení, pro konkrétní problematiku, či chce mít kompletní kontrolu nad tím, jak se data ukládají, a kde jsou uložena. (5)

2.3 Systémy a aplikace určené pro ISP

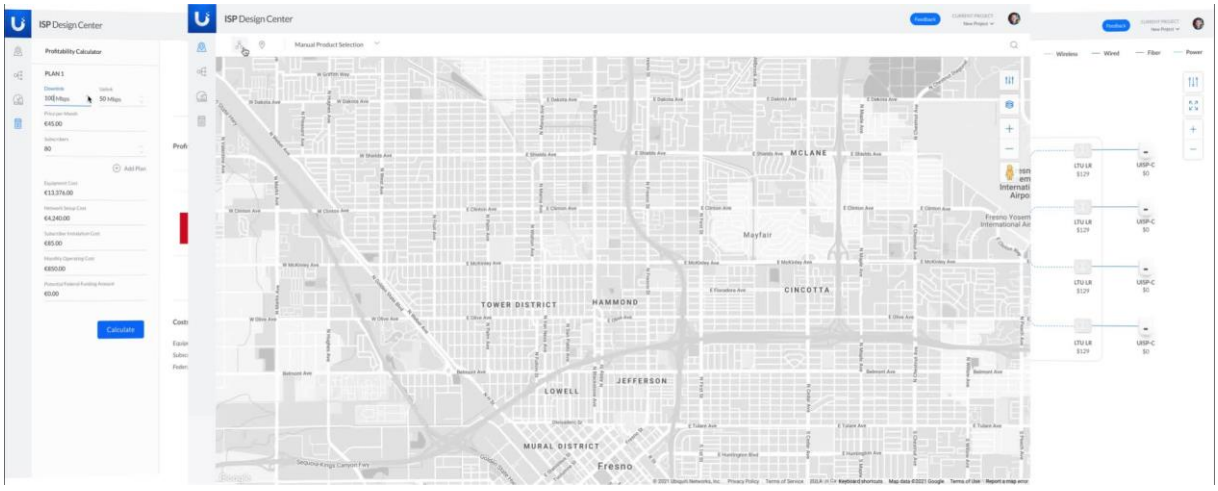
Systémů a aplikací, které jsou konkrétně zaměřené na poskytovatele, jsou v dnešní době desítky. Některé jsou velmi komplexní a nabízí řešení v podstatě na všechno, co poskytovatel potřebuje. Některé jsou zase naopak jednodušší a nabízí řešení pouze pro něco. Níže budou představeny pouze některé z nich.

2.3.1 Ubiquiti UISP

Americký výrobce Ubiquiti nabízí aplikaci UISP, kterou si poskytovatel nasadí na vlastní webový server či na jejich cloud. Tento nástroj umožňuje vykreslení topologie sítě, navrhování bezdrátových spojů a konfiguraci síťových zařízení výhradně od Ubiquiti. Do systému jdou poté přidat i zařízení, které nejsou přímo od výrobce, ale informace o těchto zařízeních, který si systém sám automaticky zjistí pomocí protokolů jako SNMP (Simple Network Management Protocol – protokol, který průběžně sbírá data pro potřeby správy sítě) jsou omezené. Obecně největší silou této společnosti je jejich velmi uživatelsky přívětivé softwarové prostředí. Systém je velmi jednoduchý a přehledný, a to zároveň v kombinaci s poměrně širokými možnostmi. Největší nevýhodou, jak už bylo zmíněno je to, jak je systém vázaný na výrobce, a také pevně dané možnosti pro správu. Pokud poskytovatel využívá jakékoliv produkty výrobce není na škodu využít i tento systém. (6)

Unifi

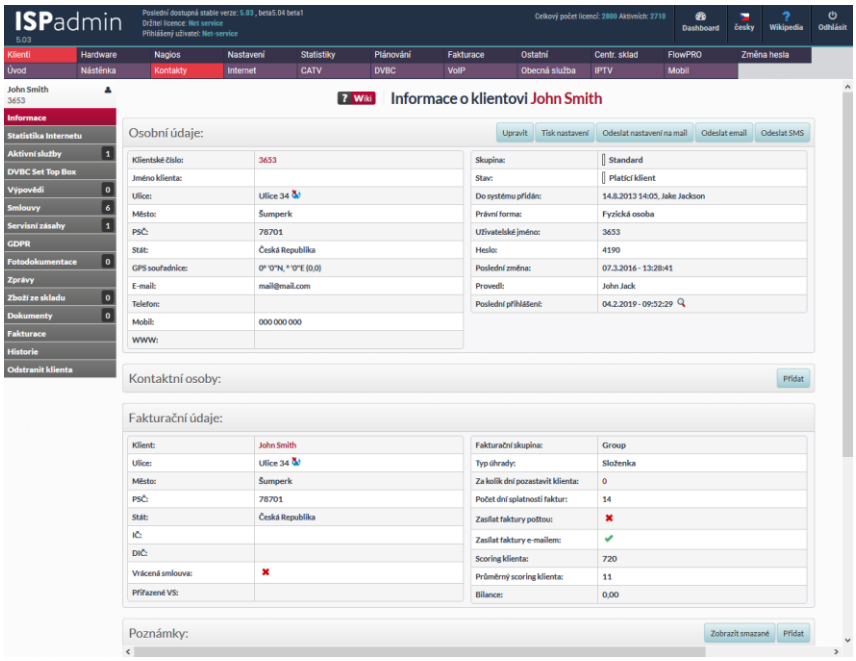
Dále výrobce nabízí aplikaci pro správu jejich wifi a jiných zařízení v lokální síti. Největší výhodou je to, že aplikace může běžet online a lokální síť může být spravována vzdáleně. To zároveň přináší největší nevýhodu, kterou je potřeba zabezpečit přístup ke správě, aby toho případný útočník nemohl zneužít Velkou výhodou využití takovéto aplikace je velké zjednodušení správy a jednoduché nasazení. Poskytovatel tuto aplikaci může využít například pro správu klientských wifi a lokální sítě formou služby. (7)



Obrázek 1 – Náhled aplikace UISP (7)

2.3.2 ISPadmin

ISPadmin je komplexní softwarové řešení pro poskytovatele internetu české firmy NET service solution. V rámci práce poskytovatele internetu nabízí práci s hardwarem, záznamy klientů a služeb, monitoring a management sítě, helpdesk pro klienty, podporu od výrobce a mnohem více. Dále nabízí také plánování, fakturaci a správu skladu. Aplikace je webová a je možné ji nasadit na vlastní server či na cloud. Systém je modulární a to, zda se využijí dané možnosti záleží čistě na potřebách poskytovatele. Od toho se poté odvíjí cena, kdy se za každého klienta v systému platí měsíční poplatek u každého využívaného modulu. To znamená, že čím více modulů bude poskytovatel využívat, tím dražší pro něj systém bude, a to samé platí u počtu klientů. Tedy čím více klientů a modulů, tím větší měsíční poplatek. (8)



Obrázek 2 – Náhled aplikace ISP admin (8)

2.3.3 DCImanager

DCImanager je jedno ze softwarových řešení společnosti ISPsystem. Jedná se o platformu, která umožňuje správu a poskytování dedikovaných serverů a jiné IT infrastruktury, skrze jedno rozhraní. Tato platforma neslouží přímo k poskytování internetu, ale je možné ji použít pro rozšíření služeb poskytovatele. Nabízí jednotnou správu IT infrastruktury, inventarizaci této infrastruktury, API (Application Programming Interface), základní správa síťových zařízení, monitoring, a tak podobně. Hlavní výhodou je ale možnost poskytovat

klientům přístup k jejich pronajaté infrastruktuře skrze webové stránky, takže se o to nemusí poskytovatel starat. (9) (10)

3 Relační databáze

Relační databáze je organizované úložiště dat v počítačovém systému, které ukládá data do sloupců a řádků a tvoří z nich tabulky. Každý řádek v tabulce relační databáze má svůj jedinečný identifikátor zvaný primární klíč. Poté má každá tabulka atributy (sloupce), do kterých se vyplňují potřebná data na každém řádku. Data jsou obvykle strukturována napříč více tabulek, které lze propojit pomocí vazeb (relací). Tyto vazby jsou definovány nejčastěji pomocí zmiňovaného primárního a cizího klíče. Kdy cizí klíč označuje primární klíč nebo jiný unikátní jednoznačný ukazatel jiného záznamu (kandidátního klíče) v databázi. Tímto způsobem vzniká vazba. Tyto vazby mezi tabulkami nemusí být povinné. (11) (12)

3.1 Entity a atributy

Entita je nějaký objekt a v relačních databázích nejčastěji tabulka. Tělo entity je tvořeno jejími atributy (sloupci), které ji definují. Každý atribut má určený svůj datový typ a to, zda je povinný či ne. (13)

Primární klíč

Jednoznačný identifikátor řádku v tabulce. Který je tvořený z jednoho nebo z kombinace nenulových atributů (sloupců). Tabulka nemůže mít více primárních klíčů, ale primární klíč mít musí. (14)

Kandidátní klíč

Jednoznačný identifikátor záznamu v tabulce. Každý záznam musí mít tento atribut (sloupec) unikátní. Každý primární klíč je kandidátní klíč, ale ne naopak. Může to být např. MAC adresa, kterou má každé zařízení unikátní. (15)

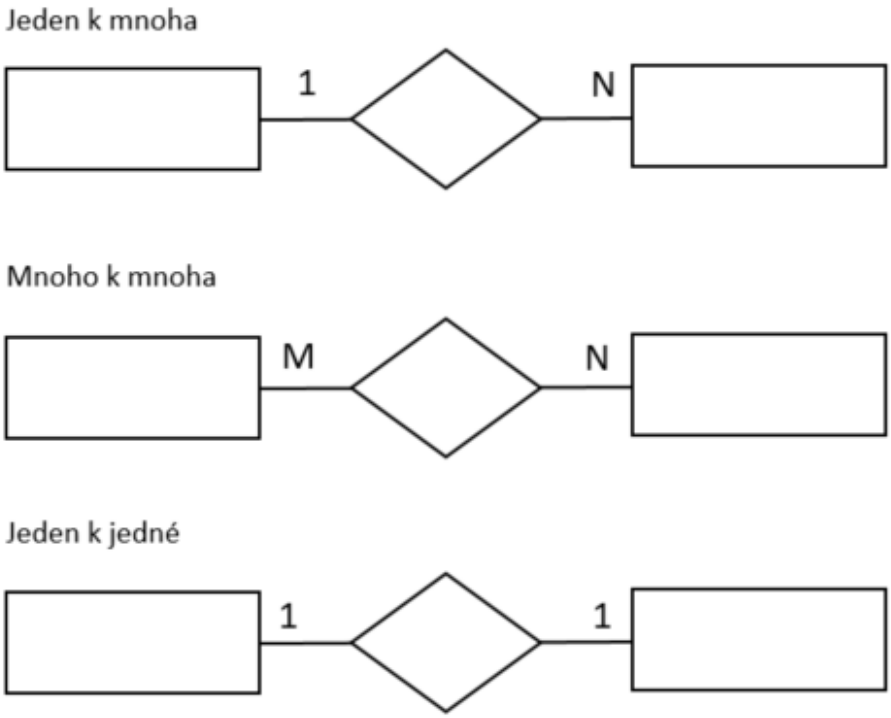
Cizí klíč

Sloupec nebo skupina sloupců v tabulce, které zajišťují spojení mezi dvěma tabulkami. Funguje jako odkaz na jinou tabulku, protože odkazuje na primární klíč jiné tabulky, čímž mezi nimi vytváří vazbu. (16)

3.2 Vztahy

Vztahy definují, jak jsou entity navzájem spojeny nebo jak spolu souvisí. Vztah se poté nadále definuje pomocí kardinality a parciality. Kardinalita říká, kolikrát se může instance dané

entity účastnit vztahu s instancemi druhé entity. Definuje tedy vztahy mezi dvěma entitami nebo množinou entit. Existují 3 hlavní typy kardinalit (Obrázek 3).



Obrázek 3 – Typy kardinalit (13)

Parcialita poté definuje, zda je vztah povinný či nepovinný. Což se definuje pomocí toho, zda může být cizí klíč nulový nebo ne. Jinak řečeno kardinalita určuje maximální počet vztahů a parcialita minimální počet vztahů. (13)

3.3 Normalizace

Normalizace je proces, ve kterém se snažíme minimalizovat redundanci dat. Tedy snažíme se, aby se data neopakovala. Redundance může zapříčinit nesprávný chod databáze, proto se využívají normální formy, aby se redundance eliminovala. Někdy je však pro správný chod aplikace postavené nad databází redundance nutná, proto by měla být normalizace vždy specifická pro potřeby použití dané databáze. (17)

3.3.1 Normální formy

Normální formy jsou skupina pravidel využívaných pro normalizaci databáze, aby se zajistil její správný chod a funkcionalita. Formy jsou rozdělené dle číslic a vyšší normální

forma, by vždy měla splňovat vše co definuje předešlá normální forma. Níže si představíme normální formy, které se týkají vztahů primárního klíče a neklíčových atributů. (17)

První normální forma

Nejzákladnější úroveň normalizace. První normální forma říká, že každý atribut musí obsahovat pouze atomické hodnoty. A každý sloupec by měl mít v rámci tabulky unikátní název. (17)

Druhá normální forma

2 NF říká, že každý neklíčový atribut musí být závislí na primárním klíči. To znamená, že každý sloupec by měl být přímo závislý na primárním klíči. (17)

Třetí normální forma

3 NF říká, že neklíčové atributy na sobě nesmí být vzájemně závislé. To opět znamená, že všechny neklíčové atributy by měli být závislé na (pouze) primárním klíči. (17)

Boyce-Coddova normální forma

BCNF říká, že neklíčové atributy jsou závislé pouze na kandidátním klíči. A že atributy (pokud je jich více), které tvoří primární klíč jsou na sobě nezávislé. (17) (18)

3.4 SQL

SQL (Structured Query Language) je programovací jazyk, který byl vytvořený pro práci s daty v relačních databázích. To znamená, že pracuje s informacemi v tabulkové struktuře. SQL se používá pro ukládání, přidávání, upravování, vyhledávání a získávání informací z tabulek v databázi.

Jedná se o nejpopulárnější jazyk pro práci s daty napříč databázemi a dalšími aplikacemi. SQL se používá napříč všemi programovacími jazyky a je důležitou částí portfolia každého, kdo se pohybuje ve vyšší sféře informačních technologií. (19)

3.4.1 Funkcionalita SQL

Vždy někde musíte mít infrastrukturu, kde může být databázový server spuštěn a na kterém se data budou ukládat. Na tento server se poté posílají SQL dotazy, které server zpracovává a vrací výsledky pro daný dotaz (odpověď). Proces probíhá skrze několik softwarových komponent serveru, které si níže představíme. (19)

Parser

Tokenizace neboli nahrazení některých slov v SQL příkazu speciálními symboly. Poté proběhne kontrola správnosti a autorizace. Kontroluje, zda příkaz odpovídá sémantice jazyka SQL. A pokud neodpovídá, tak vrátí jak odpověď chybu. Dále ověřuje, zda uživatel, který dotaz spouští má oprávnění k dané manipulaci s danými daty. (19)

Relational engine (Query processor)

Procesor dotazů, který vytvoří plán, jak se bude dotaz postupně realizovat, aby to byl co nejefektivnější způsob pro provedení daného dotazu. (19)

Storage engine (Database engine)

Softwarový komponent, který zpracovává samotný definovaný plán. Přímo zapisuje a čte data z fyzických disků. Jakmile vše dokončí vrátí potřebnou odpověď. (19)

3.4.2 SQL příkazy (dotazy)

Jedná se o příkazy nebo specifická definovaná klíčová slova, které uživatelé databáze používají pro manipulaci s daty uloženými v relační databázi. SQL příkazy jsou poté kategorizovány do různých skupin. (20)

Data Definiton Language

DDL je skupina příkazů, které se používají k definici samotné databáze. Používá se k manipulaci se samotnou databází. Jako například vytváření, mazání, vyprázdnění a tak dále. Patří sem příkazy jako CREATE a DROP. (20)

Data Query Language

Tato skupina se skládá z jediného příkazu a tím je SELECT. Tento příkaz umožňuje pouze získání dat z už definovaných tabulek v databázi. Takže skupina se dá jednoduše popsat jako operace pro čtení. (20)

Data Manipulation Language

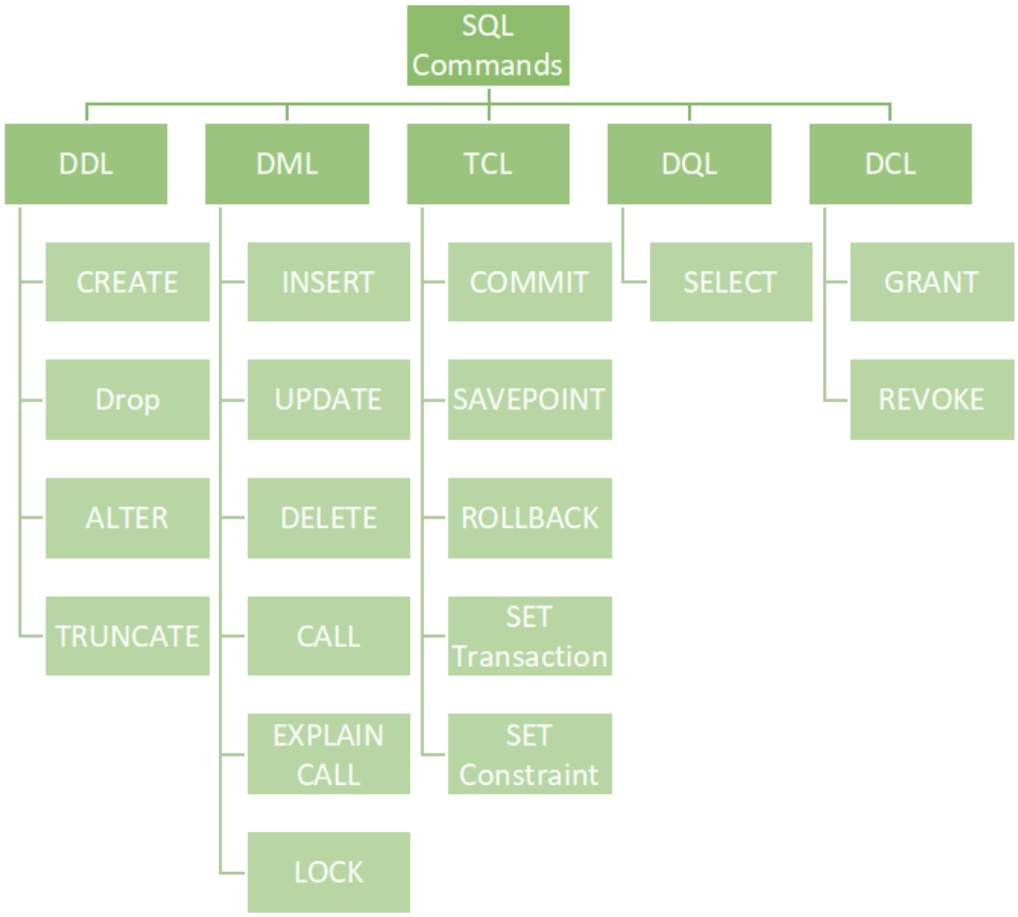
Obsahuje příkazy, které umožňují manipulovat s daty. Jako je například přidávání, upravování a mazání. Patří sem příkazy jako INSERT, UPDATE a DELETE. (20)

Data Control Language

DCL obsahuje příkazy, které se týkají oprávnění uživatelů a dalšími kontrolami v databázi. DCL a předchozí DML se často slučují do jedné skupiny příkazů. Patřím sem příkazy jako GRANT a REVOKE. (20)

Transaction Control Language

Skupina příkazů TCL obsahuje příkazy, které definují tzv. transakci. Transakce seskupuje úlohy do jedné sady úloh, které jsou pak prováděny společně. Každá transakce začíná a končí konkrétní úlohou a jakmile jsou všechny úlohy dokončeny, tak transakce vrátí výsledek. Pokud některá z úloh selže transakce se nezdaří. Transakce má tedy dva výsledky, kterými jsou buď úspěch nebo neúspěch. Patří sem příkazy jako COMMIT a ROLLBACK. (20) (21)



Obrázek 4 – SQL příkazy (20)

3.5 Indexování a optimalizace

Index je kopie části tabulky uspořádaná tak, aby umožnila databázi rychle vyhledat a načíst řádky, které odpovídají dané podmínce. Při provádění dotazu se databázový systém podívá na dostupné indexy a zjistí, zda lze některý z nich použít pro splnění podmínky dotazu. Pokud systém najde potřebný index, tak ho použije pro rychlou identifikaci odpovídajícího

řádku v dané podmínce. Díky tomu dojde k razantnímu zrychlení dotazů, a to hlavně ve chvíli, kdy je hodně dat v tabulce a podmínky jsou rozsáhlé. O tom, jestli databáze vůbec využije existujících indexů už záleží na procesoru dotazů, který vytváří plán. Ten si totiž určí, zda se to vůbec časově vyplatí.

Ty nejdůležitější indexy se ve většině databázových systému vytváří samy. A těmi jsou nejčastěji indexy pro primární klíče a unikátní sloupce v tabulce. Je tomu tak díky tomu, že pomocí těchto sloupců dokážeme identifikovat jednotlivé záznamy v tabulce. Tím pádem se nejčastěji využívají v podmínkách a databázový systém pro ně vytvoří indexy automaticky. Pak se většinou ještě vytváří indexy pro cizí klíče a sloupce, které se velmi často používají v podmínkách. To už ale záleží na konkrétní situaci.

Indexy by se měli pravidelně aktualizovat, aby odpovídal změnám v daných tabulkách. Hlavní je, aby se indexy aktualizovaly při modifikacích jako je přidávání, upravování a mazání. To ale v dnešní době už databázové systémy jako PostgreSQL řeší automaticky. (22) (23)

3.5.1 Typy indexů

B-tree index je nejčastěji používaným typem indexu pro efektivní ukládání a vyhledávání dat. Index b-tree je uspořádán do stromové struktury. Struktura začíná kořenovým uzlem (root node), která potom ukazuje na své podřízené (child) uzly. Každý uzel poté obsahuje několik dvojic, které jsou ve smyslu klíč a hodnota, kde klíč slouží k indexování a hodnota ukazuje na samotná data v tabulce. Tento typ indexu bývá základní, takže když se při vytváření indexu nespecifikuje typ vytvoří se právě tento.

Hash indexy jsou určeny pro rychlé vyhledávání klíčových hodnot. Nejčastěji se používají, pokud se u indexovaného sloupce kontroluje rovnost. Hash funkce totiž ukazuje přímo na místo, kde jsou data uloženy. Jsou o něco rychlejší než základní b-tree indexy, ale udržování aktuality těchto indexů je naopak mnohem náročnější, proto se většinu času stejně využívají b-tree indexy.

Indexy GIST (Generalized Search Tree) a SP-GIST (Space-Partitioned GIST) jsou pokročilé druhy indexů v PostgreSQL databázích, které podporují širokou škálu datových typů a operací pro vyhledávání. Nejčastěji se tyto typy indexů využívají při fulltextovém vyhledávání.

BRIN (Block Range Index) je typ indexu v PostgreSQL určený pro efektivní indexování velkých tabulek. Zejména pak velkých tabulek, které jsou nějakým způsobem pravidelně tříděné (uspořádané), či vykazují nějaký sekvenční charakteristiky. (22)

3.5.2 Caching

Dále existuje mnoho nástrojů, které databázový systém využívá sám od sebe, aby optimalizoval dotazy. Jeden z těchto nástrojů je tzv. caching. To znamená, že si databáze dočasně ukládá data, které se často používají a často se k nim přistupuje. Díky tomu dokáže opakující se dotazy zpracovat několikanásobně rychleji. Caching většinou neprobíhá pouze přímo v databázi, ale i na aplikační vrstvě. Mnoho aplikačních nástrojů si totiž také ukládá opakující se data a využívá různých technik, aby se práce s daty zrychlila. (22) (23)

3.5.3 Limitace, selekce a další

Limitací se převážně myslí využití techniky stránkování. To znamená, že databáze nemusí vracet a zpracovávat všechny záznamy naráz, ale může data zpracovávat po menších lépe zvladatelných částech. Díky tomu, jsou poté výsledky přehlednější a samotný dotaz je rychlejší.

Pro každý dotaz, kde jsou potřebná jen určitá data by se měla aplikovat selekce. Tím je myšleno, že určí přesně ty sloupce, které se mají v odpovědi na dotaz vrátit. Tím se opět omezí počet dat, s kterými musí databáze manipulovat, a také se tím samozřejmě zrychlí dotaz.

Dále existuje mnoho dalších technik, které jsou všechny vhodné pro konkrétní využití. To je například pravidelné obnovování statistik databáze nebo výběru správného způsobu pro spojení tabulek v rámci dotazu. (22) (23)

4 PostgreSQL

PostgreSQL (známé také jako Postgres) je výkonný open-source objektový a relační databázový systém, který je v aktivním vývoji už přes 35 let. Komunita vývojářů a přispěvatelů, která vyvíjí a posouvá tento databázový systém, známá jako PostgreSQL Global Development Group dotáhla Postgres do podoby, v jaké je dnes a stále pravidelně vylepšují a přidávají nové funkce. Postgres využívá a rozšiřuje SQL v kombinaci s dalšími funkcemi, které pomáhají bezpečně ukládat data, napomáhat v práci s daty a zajistit jejich integritu.

Tento databázový systém přichází s mnoha funkcemi, jejichž cílem je usnadnit zlepšit vývoj aplikací, ochranu integrity dat a pomoct se správou dat bez ohledu na jejich množství. Dále umožňuje definovat vlastní funkce, procedury, spouštěče, datové typy a tak dále. (24) (25)

4.1 Výhody použití PostgreSQL

PostgreSQL má výhradně bohatou historii podpory široké škály datových typů. A velmi kvalitní podporu optimalizace, která je běžná hlavně mezi konkurenty jako je Oracle, který je ale na rozdíl od Postgres placený. (24) (25)

4.1.1 Nabídka funkcí a rozšíření

PostgreSQL nabízí širokou a robustní škálu funkcí včetně asynchronní replikace, vnořených transakcích, vylepšený plánovače/optimalizátoru dotazů a ještě více. Dále komunita vyvinula rozšíření, které např. umožňují podporu programovacích jazyků jako Python, Java, Go, C/C++ a další, nebo umožňují ukládání velkých binárních souborů jako jsou obrázky, videa, mapy a tak dále.

Systém je velmi dobře škálovatelný jak ve směru kvantity dat, tak ve směru počtu uživatelů, kteří do ní přistupují naráz. (24) (25)

4.1.2 Tolerance chybovosti a čitelnost

PostgreSQL je kompatibilní s ACID a je vysoce odolná vůči chybám. Zkratka ACID stojí za čtyřmi vlastnostmi databázových transakcí, které je dělají spolehlivými. Jednotlivá písmena odkazují na tyto vlastnosti: atomicita, konzistence, izolovanost a trvalost. Zjednodušeně ACID zajišťuje nebo spíše označuje, že data v databázi jsou přesná a úplná, protože neúplné změny se nikdy neuloží do databáze, a to například ve chvíli, kdy nastane

chyba během transakce. Díky tomu Postgres poskytuje vyšší integritu dat než jiné SQL databáze. (24) (25)

4.1.3 Open-source a komunita

Zdrojový kód systému je dostupný pod open-source licencí, která umožňuje bezplatné použití a úprava systému, dle potřeby. Díky tomu se Postgres používá i ve velkých firmách, kde si databázový systém upravují dle potřeby.

Díky oddané komunitě se vývoj databázového systému nezastavuje a pokračuje stále kupředu. Další výhodou rozšířené a oddané komunity je poté podpora a pomoc ze strany komunity, která nabízí velké množství návodů a rad pro používání a vylepšování. (24) (25)

4.2 Procedury, pohledy a spouštěče

Procedury a funkce umožňují v PostgreSQL spustit sadu operací. Obecný rozdíl mezi funkcí a procedurou je pak to, že funkce vždy vrací nějaký výsledek. Procedura pouze splní sadu operací. Hlavním rozdíl v rámci Postgres je to, že funkce neumožňuje použití TCL.

Pohled je něco jako virtuální tabulka postavená na výsledcích předefinovaného dotazu. Může obsahovat data z jedné nebo více tabulek. Postgres navíc obsahuje nástroje jako předsestavené pohledy, což znamená, že se dotaz v rámci pohledu provádí asynchronně s dotazem pro získání daného pohledu. Je to tedy zjednodušeně řečeno tabulka, která má duplikovaná data.

Spouštěč neboli trigger je funkce, která se spustí při specifikované události. Takže můžeme například vytvořit spouštěč, aby aktualizoval časové razítko (timestamp) při aktualizaci záznamu. (26)

4.3 Rozdíly PostgreSQL oproti jiným relačním databázím

PostgreSQL je na rozdíl od jiných relačních databází kombinací relační a objektové databáze. To znamená uložení dat do tabulek a vytváření vazeb mezi nimi v kombinaci s funkcemi objektové databází, což konkrétně u PostgreSQL znamená například:

- Komunikace s databázovým serverem pomocí používání objektů v kódu
- Definice vlastních komplexních datových typů
- Definice funkcí, které pracují s vlastními datovými typy
- Definice parent-child vztahů, nebo dědění, mezi tabulkami

- Datové typy jako je JSON, XML, HSTORE a Array

Návrh PostgreSQL je tedy flexibilnější než u jiných databázových serverů. Definice různých vztahů a typů se pak může definovat už přímo v databázi, a ne až v kódu aplikace. Díky tomu se pak výkon databáze může hodně zlepšit. (24) (25)

4.3.1 Porovnání PostgreSQL a MySQL

MySQL je čistě relační databázoví systém, který umožňuje ukládání dat do řádků a sloupců v tabulce a je velmi populární při tvorbě webových aplikací. PostgreSQL je objektově relační databázový systém, a díky tomu nabízí více funkcí než MySQL. (27) (28)

Podobnosti

Oba systémy ukládají data do řádků a sloupců tabulek a mezi tabulkami definují vazby, a to pomocí vzdáleného klíče, který nejčastěji odkazuje na primární klíč jiné tabulky. Dále oba systémy využívají jazyk SQL jako prostředek pro čtení a modifikaci dat v databázi a využívají dalších funkcí které SQL nabízí. Oba jsou open-source a mají velkou komunitu, která se daným systémem zabývá. (27) (28)

Rozdíly

PostgreSQL je preferováno pro optimálnější výkon při read-write operacích, nebo práci s velkým objemem dat. MySQL je naopak preferováno při read-only operacích. To je způsobeno také tím, že PostgreSQL nabízí více funkcí, tím pádem může být MySQL v nějakých případech rychlejší, lehčí a stabilnější. Některé z těchto PostgreSQL funkcí jsou například rozšířené možnosti u procedur, pohledů, spouštěčů a hlavně indexů.

PostgreSQL je kompatibilní s ACID ve všech konfiguracích, takže zajišťuje integritu dat pokaždé bez potřeby použití softwarových modulů nebo jiného formátu uložení dat. Dále PostgreSQL podporuje funkci MVCC (MultiVersion Concurrency Control), což je pokročilá funkcionalita databáze, která při modifikaci záznamu vytváří duplikát daného záznamu, aby se mohla data číst a modifikovat paraelně. Zároveň může více uživatelů číst a upravovat stejná data, aniž by byla narušena integrita dat. MySQL sice tyto funkcionality nabízí také (v případě použití určitých formátů úložiště dat), ale Postgres bylo od začátku vyvíjeno, aby je podporovalo. Tudíž ve chvíli, kdy je MVCC nezbytné, je lepší zvolit PostgreSQL.

MySQL je vysoce kompatibilní s různými typy formátu pro ukládání dat. PostgreSQL zase oproti tomu podporuje vysokou škálu datových typů a formátů, které se používají např. u NoSQL (databázové systémy, které nevyužívají jazyk SQL). (27) (28) (29)

5 Webové technologie

Webové technologie se skládají z mnoha technik, nástrojů a protokolů, které umožňují proces komunikace napříč internetem. Hlavně prostředek pro přístup a komunikace je u webových technologií prohlížeč. To je program, který umožňuje zobrazovat různé typy médií, které se nacházejí na internetu a také se na internetu pohybovat díky hypertextovým odkazům. Některé z technologií, nástrojů a protokolů budou představeny níže v podkapitolách. (30)

5.1 HTTP

HTTP (Hypertext Transfer Protocol) je základem jakékoliv výměny dat přes web, a to nejčastěji HTML dokumentů, který se většinou skládá z menších médií a dokumentů. Jedná se o aplikační TCP/IP protokol běžící na portu 80 na bázi klient-server, což znamená, že komunikace začíná dotazem klienta na server, který poté odpovídá a vrací data požadována klientem. Formu klienta většinou zastává webový prohlížeč nebo jiný user-agent, který komunikuje se serverem jménem uživatele. (31) (32) (33)

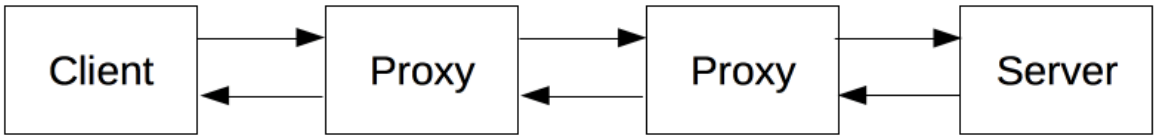
5.1.1 Funkcionalita HTTP

Pro zobrazení stránky, prohlížeč pošle dotaz pro získání HTML dokumentu, který reprezentuje webovou stránku. Tento soubor pak prohlížeč zpracuje a zažádá si o další soubory, které jsou potřeba pro zobrazit, jako jsou například CSS soubory nebo multimédia. Tyto soubory poté zkombinuje a zobrazí finální webovou stránku.

Webová stránka je tzv. hypertextový dokument. To znamená, že některé elementy na stránce jsou odkazy, přesněji hypertextové odkazy. Na které lze kliknout a přesunout se na jinou webovou stránku. Při kliknutí prohlížeč zašle HTTP dotaz na cílový soubor a přemístí uživatele.

Na opačné straně komunikace se poté nachází webový server, který zašle klientovi dokumenty, o které žádal. Pro klienta se server jeví jako jeden stroj, ale tak to být vždy nemusí. Může se jednat o skupinu webových, databázových a dalších serverů.

V komunikaci mezi klientem a serverem se také může nacházet tzv. proxy. Jedná se server, který je prostředníkem v komunikaci mezi klientem a serverem. Nejčastěji slouží pro rozložení zátěže mezi servery, autentizaci, šifrování, filtrování či logování. Komunikace z obou stran, ale probíhá jako by tam žádná proxy nebyla (Obrázek 5). (31) (32) (33)



Obrázek 5 – HTTP komunikace s proxy (31)

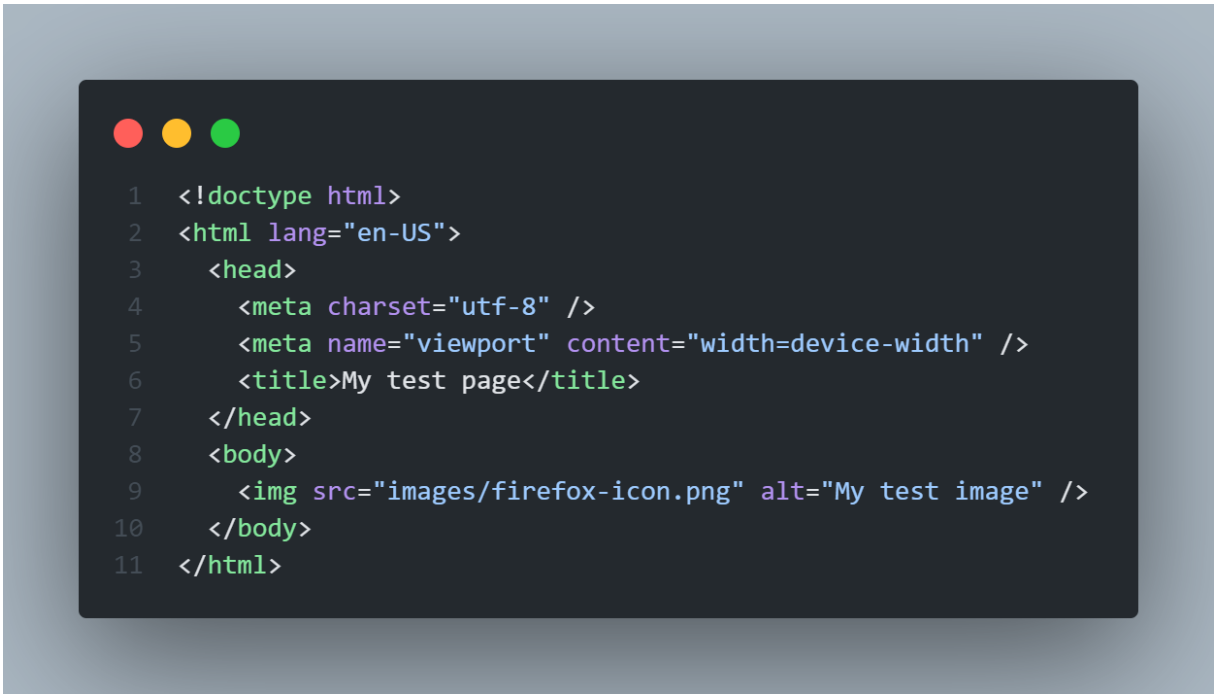
5.1.2 HTTPS

HTTPS (HTTP Secure) je nadstavbou nad HTTP, která běží na portu 443 a šifruje komunikaci pomocí SSL (Secure Sockets Layer) nebo TLS (Transport Layer Security) certifikátů. HTTPS poté šifruje a dešifruje jak žádosti, tak odpovědi (tedy celou komunikaci). V dnešní době už většina webů funguje čistě na HTTPS, protože čisté HTTP už je velmi nebezpečné a zastaralé. (31) (32)

SSL a TLS jsou protokoly pro zabezpečení komunikace pomocí šifrování. SSL a TLS se běžně moc nerozlišují, protože SSL je v podstatě jen předchůdce modernějšího protokolu TLS, a proto se jejich názvy zaměňují. TLS ve své podstatě jen opravuje chyby a zranitelnosti staršího SSL. Jinak jsou protokoly stejné. To, že je komunikace šifrovaná, znamená, že pokud by někdo zachytil komunikaci mezi klientem a serverem, získal by pouze směs znaků, která se špatně dešifruje. Dále certifikáty zajišťují integritu dat, tedy jistotu, že data v komunikaci nikdo nepoupravil a autentifikace v rámci komunikace. (31) (32) (34)

5.2 HTML a CSS

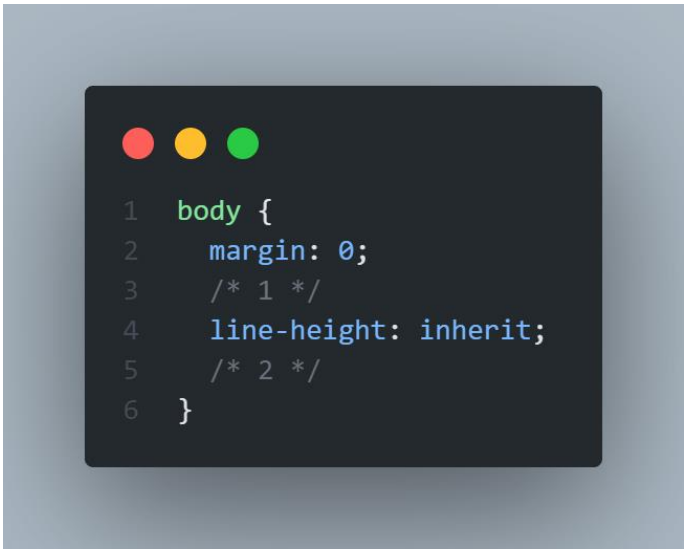
HTML (HyperText Markup Language) je jazyk, který je základním stavebním kamenem webových stránek a jejich obsahu. HTML je značkovací (tagovací) jazyk, který definuje strukturu obsahu. Skládá se z řady prvků, pomocí kterých se strukturuje všechen její obsah, tyto prvky se nazývají tagy. Tagy existují v podstatě pro všechno, a to například odstavce, multimédia a hypertextové odkazy. Jednotlivé tagy poté obsahují vlastní atributy, tedy jejich vlastnosti. Tyto vlastnosti mohou být buďto předem definované, nebo vlastní. Kdy se pak tyto atributy mohou použít v rámci např. JavaScriptu. (35)



```html
<!doctype html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>My test page</title>
  </head>
  <body>
    <img src="images/firefox-icon.png" alt="My test image" />
  </body>
</html>
```

Obrázek 6 – příklad HTML (35)

CSS (Cascading Style Sheets) je jazyk, který umožňuje stylovat a vzhledově upravovat uživatelské prostředí webové stránky. CSS lze použít pro velmi jednoduché základní stylování, jako například změnu barev a velikosti objektů. Dále také pro kompletní rozvržení dokumentu, vytváření efektů, animací a dalších pokročilých technik. (36)



```css
body {
  margin: 0;
  /* 1 */
  line-height: inherit;
  /* 2 */
}
```

Obrázek 7 – příklad CSS

5.3 PHP

PHP (Hypertext Preprocessor) je open-source server-side (back-end) skriptovací jazyk, který je využíván pro psaní webových stránek. Server-side znamená, že běží na serveru zároveň s webovým serverem a na serveru také zpracovává všechna data. Díky tomu, že je PHP ve vývoji už více než 15 let má spousty svých silných stránek. Některé z nich jsou například jednoduchá syntaxe a možnost práce se skoro jakoukoliv databází. I přesto, že je PHP tak starý jazyk, tak není ani zdaleka mrtvý díky Wordpressu, OOP, frameworkům jako je Laravel a stálému vývoji je pořád skoro 80 % webových stránek založeno právě na PHP.

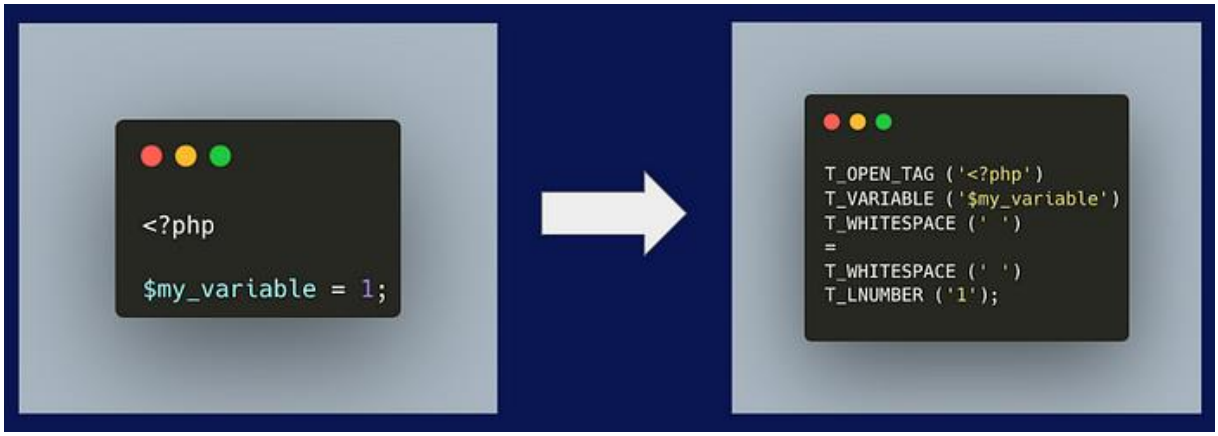
PHP je interpretovaný jazyk. To znamená, že je kód spouštěný a provádí se řádek po řádku ze shora dolů. Díky tomu je jazyk více flexibilní, a tím pádem je i mnohem jednodušší psát v PHP nepřehledně. Dále je PHP také nezávislé na platformě, na které je zprovozněno. (37) (38)

5.3.1 Jak funguje PHP

PHP kód zjednodušeně prochází čtyřmi kroky a těmi jsou: tokenizace, parsování, kompilace a jako poslední spuštění. (37) (38)

Tokenizace

Tokenizace je krok, ve kterém PHP zkontroluje zdrojový kód znaku po znaku. Dle kódu vytvoří tokeny, které jsou ve své podstatě zjednodušený postup pro další krok. Dále v tomto kroku smaže nepotřebné znaky jako například komentáře (Obrázek 8). (37) (38)



Obrázek 8 – Tokenizace PHP (38)

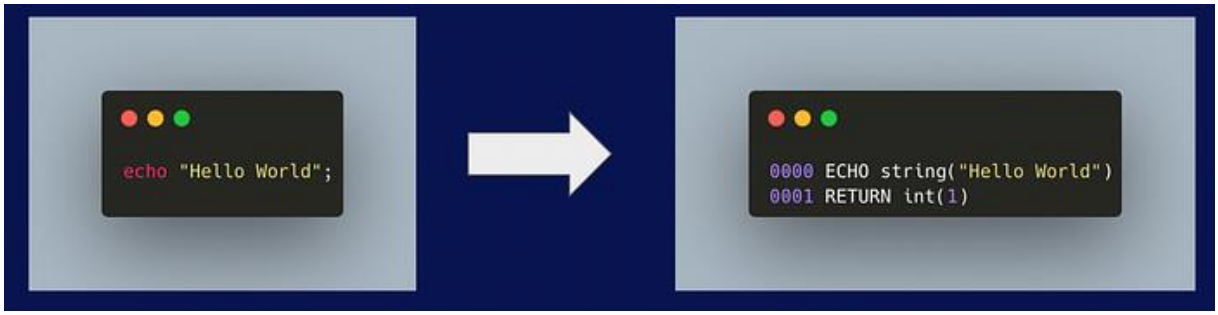
Parsování

V tomto kroku se kontroluje syntaxe. Kontroluje, zda kód odpovídá daným pravidlům PHP. Pokud neodpovídá vrací chybu, a to ať už přímo na stránku nebo do logu (záleží na konfiguraci). Poté po použije tokeny a sestaví z nich AST (Abstract Syntax Tree) pro další krok (Obrázek 9). (37) (38)

Obrázek 9 – Parsování PHP (38)

Kompilace

V tomto kroku se vygeneruje bajtový kód pomocí AST, která vznikla při během předchozího kroku (Obrázek 10). (37) (38)



Obrázek 10 – Kompilace PHP (38)

5.3.2 PHP OOP

OOP (Object-Oriented Programming) neboli objektově orientované programování se v první podobě v PHP objevilo ve verzi 5. Klasické procedurální programování je o tvoření procedur a funkcí, které pracují s daty. Oproti tomu v OOP se vytvářejí objekty, které obsahují jak data, tak funkce. Výhod OOP je několik:

- Čistší struktura programu
- Rychleji a jednodušeji proveditelné psaní
- Kód se jednodušeji udržuje, upravuje a debuguje
- Snazší psaní univerzálního kódu, který jde využít napříč aplikacemi

Třídy (classes) a objekty jsou jedny z nejdůležitějších pojmů OOP. Ve své podstatě je to jedno a to samé. Třída je šablona pro objekt, a objekt je instance této šablony. Když se vytvoří nová instance třídy (tedy objekt), tak zdědí všechny vlastnosti a pouze poupraví některá data. Třídou může být například zaměstnanec. Každý zaměstnanec má nějaké jméno, věk, pozici a tak dále. Když se poté vytváří instance zaměstnanců, tak každý bude mít jinou kombinaci těchto údajů (Obrázek 11). Stejná alegorie lze použít i třeba u ovoce. V tomto případě je název třídy ovoce a její instance jsou např. banán, jablko nebo pomeranč. (37) (39)

```php
<?php

/**
 * Třída pro zaměstnance
 */
class Employee {

    public $name;
    public $age;
    public $position;

    public function __construct($name, $age, $position)
    {

        $this->name = $name;
        $this->age = $age;
        $this->position = $position;
    }
}

//Instance třídy (objekt)
$employee = new Employee('Vojtěch Pokorný', 19, 'IT technik');
```

Obrázek 11 – Příklad PHP OOP

5.4 JavaScript

JavaScript je skriptovací jazyk, který se využívá pro vývoj převážně front-end webových aplikací. Front-end znamená, že jde o to, co uživatel vidí ve svém prohlížeči, a to s čím interaguje. Používá se v kombinaci s jazyky HTML, CSS a poté také často s libovolným server-side jazykem (nejčastěji PHP). A je také interpretovaný stejně jako PHP. JavaScript je používán zásadně pro dynamický obsah na stránce. Dynamicky mění vzhled pomocí úpravy HTML elementů, CSS komponent a dalších.

Jednou z největší výhod JavaScriptu je to, že jeho fungování je postavené na prohlížeči přímo u uživatele, tím pádem se jedná o skvělý nástroj pro práci s API (výhradně REST API). To, že je fungování postavené na prohlížeči přímo u uživatele (zkráceně client-side), znamená, že když uživatel navštěvuje webovou stránku, která obsahuje JavaScript, všechen kód je dočasně stažen k uživateli do prohlížeče a v něm následně i spuštěn. (40)

```javascript
/**
 * Odeslání JSON na PHP soubor, pomocí POST metody
 *
 * @param {string} dir cesta k cílovýmu souboru
 * @param {object} formData objekt s definovanými daty pro POST
 * @param {function} callback funkce, která se zavolá pro dokončení
 */
function sendPostJson(dir, formData, callback) {

    var xhr = new XMLHttpRequest();
    xhr.open('POST', dir, true);

    xhr.onload = function () {

        //console.log(xhr.responseText);
        if (xhr.status >= 200 && xhr.status < 300) {

            var result = JSON.parse(xhr.responseText);
        } else {

            var result = null;
        }

        if (callback) {
            callback(result);
        }
    }

    var postData = JSON.stringify(formData);
    xhr.setRequestHeader('X-Requested-With', 'XMLHttpRequest');
    xhr.setRequestHeader('Content-Type', 'application/json');
    xhr.send(postData);
}
```

Obrázek 12 – Příklad JavaScriptu

5.4.1 JSON

JSON (JavaScript Object Notation) je jednoduchý textový formát pro výměnu dat nezávislí na jazyce. JSON definuje malou sadu pravidel pro definici a reprezentaci strukturovaných dat. (41)

Obrázek 13 – Příklad JSON struktury

5.4.2 AJAX

AJAX (Asynchronous JavaScript And XML) je technika při, které se využívá objekt prohlížeče XMLHttpRequest, pomocí kterého se JavaScript dokáže asynchronně dotázat webového serveru o data, které poté rovnou zobrazí, či jenom zpracuje. AJAX se často používá v kombinaci s PHP pro získání dat z PHP souboru nacházejícího se na serveru, aniž by se webová stránka musela znovu celá načítat. (42)

6 Technická dokumentace

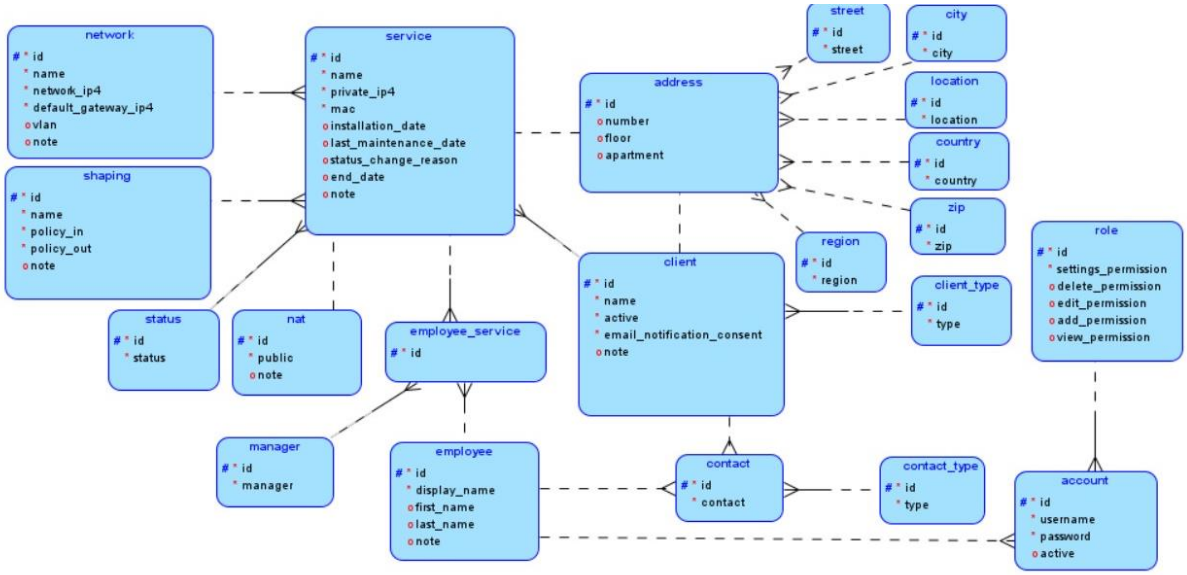
Aplikace vytvořená v rámci projektu je postavená na databázi. Zvyšuje validitu dat v databázi a umožňuje, aby s ní mohli pracovat i méně technicky schopní uživatelé. Dále přidává další funkce jako například logování či automatické emaily. Níže v této kapitole bude představen návrh a funkcionalita aplikace. Dále průběhy testování a představení použitých nástrojů.

6.1 Návrh struktury databáze

Databázový systém PostgreSQL byl vybrán výhradně díky tomu, že ve firmě Prestonet se už používá. Aby se tedy zajistila jednota v rámci firemních databází a ulehčila se jejich jednotná správa, byl tento databázoví systém zvolen i pro aplikaci. Dále se díky rozšířeným možnostem datových typů Postgres může zajistit vyšší integrita dat na čistě databázové vrstvě. Tato záležitost s výběrem databáze byla také probírána s odborníky v oboru z firmy Prestonet (https://prestonet.cz/) a Computer Systems Consulting (https://www.csc-sro.cz/).

Databáze se skládá z celkově 21 tabulek. Tabulky dále lze rozdělit na ty hlavní a na ty pomocné. Hlavními by se dali nazvat výhradně tabulky *service* a *client*, ke kterým se poté doplňují informace pomocí dalších tabulek. Níže jsou představeny tabulky a struktura pomocí logického modelu (Obrázek 14).

Včetně tabulek jsou v databázové struktuře vytvořeny indexy, spouštěče (triggery) a funkce (většinou spíše jednoduché). Dále se rozhodovalo mezi některými možnostmi pomocí testování, které je popsáno dále v projektu. Samotný skript pro vytvoření databáze se pak nachází v rámci PHP kódu v objektu *DBController* v metodě *create*.



Obrázek 14 – Logický návrh databáze

```sql
CREATE TABLE IF NOT EXISTS service(
    id SERIAL PRIMARY KEY,
    service_name VARCHAR(50) NOT NULL,
    mac MACADDR NOT NULL UNIQUE,
    private_ip4 INET NOT NULL UNIQUE,
    installation_date DATE,
    last_maintenance_date DATE,
    status_change_reason VARCHAR(100),
    end_date DATE,
    note VARCHAR(300),
    client_id INT NOT NULL,
    address_id INT UNIQUE,
    status_id INT NOT NULL,
    network_id INT NOT NULL,
    nat_id INT UNIQUE,
    shaping_id INT NOT NULL,
    CONSTRAINT fk_client FOREIGN KEY(client_id) REFERENCES client(id),
    CONSTRAINT fk_address FOREIGN KEY(address_id) REFERENCES address(id),
    CONSTRAINT fk_status FOREIGN KEY(status_id) REFERENCES status(id),
    CONSTRAINT fk_network FOREIGN KEY(network_id) REFERENCES network(id),
    CONSTRAINT fk_nat FOREIGN KEY(nat_id) REFERENCES nat(id),
    CONSTRAINT fk_shaping FOREIGN KEY(shaping_id) REFERENCES shaping(id)
);
```

Obrázek 15 – Tabulka *service*

6.1.1 Tabulky, vazby a normalizace

Nejdůležitější tabulkou v databázi je *service*. Slučuje totiž všechny potřebné technické informace o konkrétní internetové službě. Obsahuje svá základní data, a poté má vazby s mnoha tabulkami, z kterých získává další potřebná data. Tabulky jsou rozděleny kvůli přehlednosti a plnění normalizačních pravidel. Konkrétní vazby tabulky *service* a jejich popisek:

- *network (1:N)* – Každá služba se musí nacházet v síti, která má *n* služeb.
- *shaping (1:N)* – Každá služba musí mít tarif, který má *n* služeb. Určuje rychlost internetu pro danou službu (tedy shaping).
- *status (1:N)* – Každá služba musí být v nějakém stavu. Daný stav má *n* služeb. Stavy jsou předem definovány na hodnoty: V procesu připojování, Aktivní, Pozastaveno, V údržbě a Ukončeno.
- *nat (1:1)* – Služba může mít NAT 1:1, který přidělí dané službě veřejnou adresu, která bude jenom pro ni. Veřejná adresa není přímo v tabulce *service*, kvůli přehlednosti a možnosti zpětného přiřazení.
- *employee_service (N:1)* – Tato tabulka je pomocná a slouží pro propojení zaměstnanců se službami. Každá služba má v momentální verzi aplikace max. 2 manažery (Technik a Obchodník), které jsou předem definovány podobně jako stavy. Tabulka *employee_service* obsahuje klíče zaměstnance, služby a manažera. Manažer (*manager*) je také pomocná tabulka, která obsahuje právě typy manažerů, které jsou zmíněny výše. Každý záznam poté musí obsahovat unikátní kombinaci primárního klíče služby a manažera, aby se manažer nemohl opakovat.
- *client (1:N)* – Každá služba musí mít nějakého klienta, který má *n* služeb.
- *address (1:1)* – Každá služba je na své unikátní adrese, která je přímo vázaná na danou službu. Podrobnosti k této tabulce budou představeny níže. Adresa nemusí být vyplněna.

Další důležitou tabulkou je *client*. Ta slučuje důležité informace o klientovi, podobně jak to dělá tabulka *service*. Konkrétní vazby tabulky *client* a jejich popisek:

- *service (N:1)* – Každý klient má *n* služeb, která má pouze 1 klienta.
- *address (1:1)* – Každý klient má svou unikátní adresu, která je na něj přímo vázaná (adresa bydliště). Podrobnosti k této tabulce budou představeny níže. Adresa nemusí být vyplněna.
- *client_type (1:N)* – Každý klient je nějakého typu, typ má *n* klientů. Typy jsou definovány přímo v aplikace a pole slouží pro vlastní kategorizaci.
- *contact (N:1)* – Klient může mít až 4 typy kontaktů, které jsou předem definované. Jsou jimi primární/technické email/telefonní číslo.

Tabulka *address* je plně normalizována. Pole pro město, ulici, zip atd. jsou všechny odděleny do pomocných tabulek. Tabulka *address* je poté se všemi těmito tabulkami v nepovinné vazbě 1:N. Kvůli tomu, aby se nemuselo určovat, co se konkrétně vyplní pro konkrétní adresu. Všechna její pole jsou tedy nepovinná. Tudíž se vždy můžou vyplnit libovolná pole. To, aby v databázi nezůstávaly prázdné záznamy, je ohlídané přímo v databázi pomocí triggerů (Obrázek 17) (Obrázek 16). Adresa má vazby 1:1 s tabulkami *service* a *client*. Každý objekt má svoji vlastní adresu, kvůli aplikačním procesům. Bylo by totiž velmi složité kdyby např. 1 adresu mělo více klientů a jeden by se například odstěhoval. V té chvíli by se musel změnit záznam jeho adresy, ale tento záznam by měl vazby na jiné klienty, pro které stará adresa stále platí.

```
isp=# SELECT * from address where id = 1;
 id | number | floor | apartment | street_id | zip_id | city_id | region_id | country_id | location_id
----+--------+-------+-----------+-----------+--------+---------+-----------+------------+-------------
  1 | 1      | 2     | Apt 12    |        44 |     31 |      70 |        98 |         22 |          64
(1 row)

isp=# SELECT * from address where id = 1;
 id | number | floor | apartment | street_id | zip_id | city_id | region_id | country_id | location_id
----+--------+-------+-----------+-----------+--------+---------+-----------+------------+-------------
(0 rows)
```

Obrázek 17 – Adresa před/po odstranění všech polí v adrese

```
1  CREATE TRIGGER trg_check_null_address
2  AFTER INSERT OR UPDATE ON address
3  FOR EACH ROW
4  EXECUTE FUNCTION fn_delete_address_if_null();
```

Obrázek 16 – Trigger spouštějící funkci

Dále jsou tabulky *employee* a *contact*. Tabulka *employee* obsahuje zaměstnance, kteří se vážou ke službám. Jednotliví zaměstnanci se vážou, jak už bylo zmíněno, na tabulku *employee_service* a dále na kontakty. Těmito jsou primární (pracovní) email/telefonní číslo. Na tabulku *contact* je kromě zaměstnanců a klientů vázána tabulka *contact_type*. Ta není plně

normalizována, protože vzhledem k malému počtu možných kontaktů se nevyplatí ji plně normalizovat a rozdělovat *contact_type* ještě dále do více tabulek.

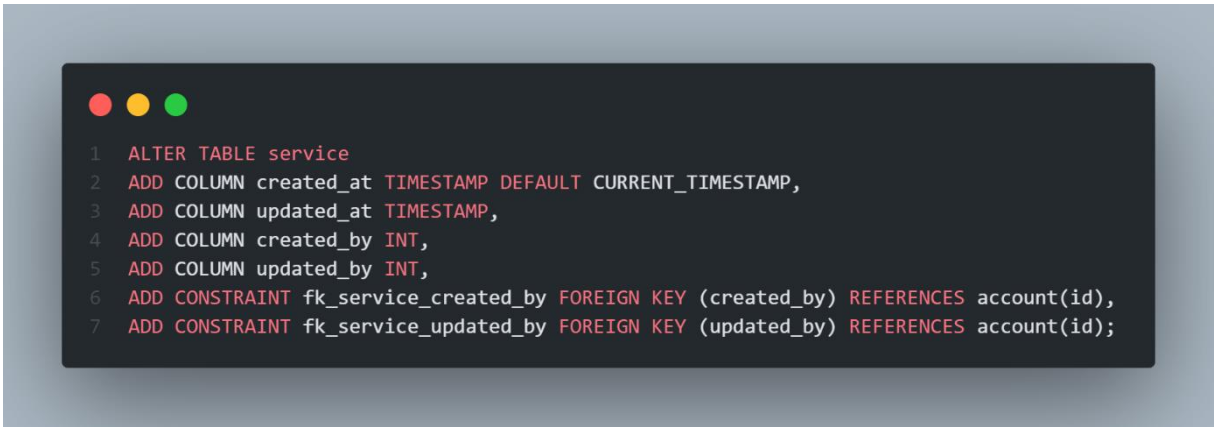
Jako poslední je tabulka *account*, která slouží pro ukládání uživatelských účtů. Konkrétní vazby tabulky *account* a jejich popisek:

- *role (1:N)* – Každý účet má přiřazenou roli, která má *n* účtů. Role obsahuje několik polí datového typu BOOLEAN, které určují, jaké má daná role oprávnění.
- *employee (1:N)* – Účet může být vázaný na 1 zaměstnance, ten může mít *n* účtů.

6.1.2 Pole tabulek

Pole v tabulkách jsou výhradně datových typů VARCHAR. Ten je určený pro zaznamenávání řetězců, které mají určenou maximální délku. Dále se často využívají datové typy INT a SERIAL, které se nejčastěji využívají pro ukládání klíčů. První (INT) se používá pro cizí i primární klíče a jedná se o obyčejné číslo. SERIAL je poté v podstatě to samé, ale jen se dané číslo automaticky zvyšuje vždy, když se přidává nový záznam. Používá se tedy ve většině tabulek pro primární klíč (*id*). Poslední dva základnáí typy, které se využívají jsou BOOLEAN a DATE. První z těchto dvou představuje jen velmi obyčejné pole, které může nabýt hodnoty true (pravda) nebo false (nepravda). To je použito například u klienta, kde se v poli *active* jednoduše určuje, zda je daný klient aktivní. DATE je poté pole pro jednoduché zaznamenání data.

Hlavní záznamy, které lze přímo upravovat v rámci aplikace mají dále (už v uživatelské dokumentaci zmíněné) analytické pole (Obrázek 18). Díky těmto polím mají v podstatě všechny tabulky vazbu s tabulkou *account*. To ale pro přehlednost není znázorněno v logickém návrhu. Pole *created_by* a *update_by* jsou cizí klíče odkazující na primární klíč tabulky



Obrázek 18 – Analytická pole

account, kde jsou uloženy všechny účty. Druhé dvě pole *created_at* a *updated_at* jsou datového typu timestamp (časové razítko). Ty se aktualizují na úrovni databáze pomocí triggeru, který se spustí vždy ve chvíli, kdy se záznam aktualizuje nebo vytvoří (Obrázek 19). Trigger je vytvořený pro každou tabulku, u které je to potřeba. Každý z nich volá funkci pro samotnou aktualizaci pole. Časové pásmo, které určuje, jaký čas bude do časového razítka vygenerován, se nastavuje při inicializaci spojení na Europe/Prague.

```
CREATE OR REPLACE FUNCTION fn_update_updated_at()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP AT TIME ZONE 'Europe/Prague';
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_updated_at_service
BEFORE UPDATE
ON service
FOR EACH ROW
EXECUTE FUNCTION fn_update_updated_at();
```

Obrázek 19 – Funkcionalita pole *updated_at*

Adresy jsou v rámci tabulek ukládány v datových typech MACADDR, INET a CIDR. Další dva datové typy slouží pro zaznamenávání IP adres v rámci PostgreSQL databáze. Datový typ INET je určený přímo pro zaznamenávání jednotlivých adres (bez masky), takže např. *private_ip4* v tabulce *service*. Typ CIDR je využíván pro zaznamenávání rozsahu adres (sítě), kde se ukládá adresa s bitovým představením masky např. 10.0.1.0/24. Tento datový typ je použit například u pole *network_ip4*. (43)

Dále pole nabývají různých parametrů, které určují další činnosti a vlastnosti pole. Nejčastěji používaný parametr je NOT NULL, ten říká, že dané pole nesmí být nulové a tím pádem je povinné ho vyplnit. Toto pole se využívá pro vyjádření parciality vztahu mezi tabulkami. Dalšími používanými parametry jsou UNIQUE a DEFAULT. UNIQUE říká, že v rámci tabulky je hodnota unikátní. Využívá se pro určení kardinality 1:1, které se nachází

například mezi tabulkami *nat* a *service*. DEFAULT určuje defaultní hodnotu pole. Toto se využívá pro základní určení oprávnění v roli, nebo pro timestamp *created_at*, kam se automaticky vyplní aktuální časové razítko.

Mezi další parametry, které se využívají v rámci tabulek, jsou PRIMARY KEY a FOREIGN KEY. Tyto parametry definují PostgreSQL databázi, zda jsou pole součástí nějakého vztahu. PRIMARY KEY se nastavuje k primárním klíčům tabulek a PostgreSQL databázi říká, že dané pole musí být nenulové a unikátní. Dále lze nastavit pro PostgreSQL explicitně, zda je pole cizím klíčem. (44)

6.1.3 Indexy

V databázi jsou použity indexy typu B-tree. Jak už bylo řečeno výše v projektu, tak primární klíče jsou vytvořeny automaticky spolu s indexy pro unikátní pole jednotlivých tabulek (tedy kandidátní klíče). Dále jsou vytvořeny indexy pro jednotlivé cizí klíče, které databáze využívá při operacích pro spojování tabulek (JOIN) (Obrázek 20). Žádné další druhy indexů využity nejsou, protože testování ukázalo, že tyto indexy (např. pro pole, které se využívají v podmínce) databáze nevyužívá a v případě vynucení jejich použití je získání dat pomalejší.

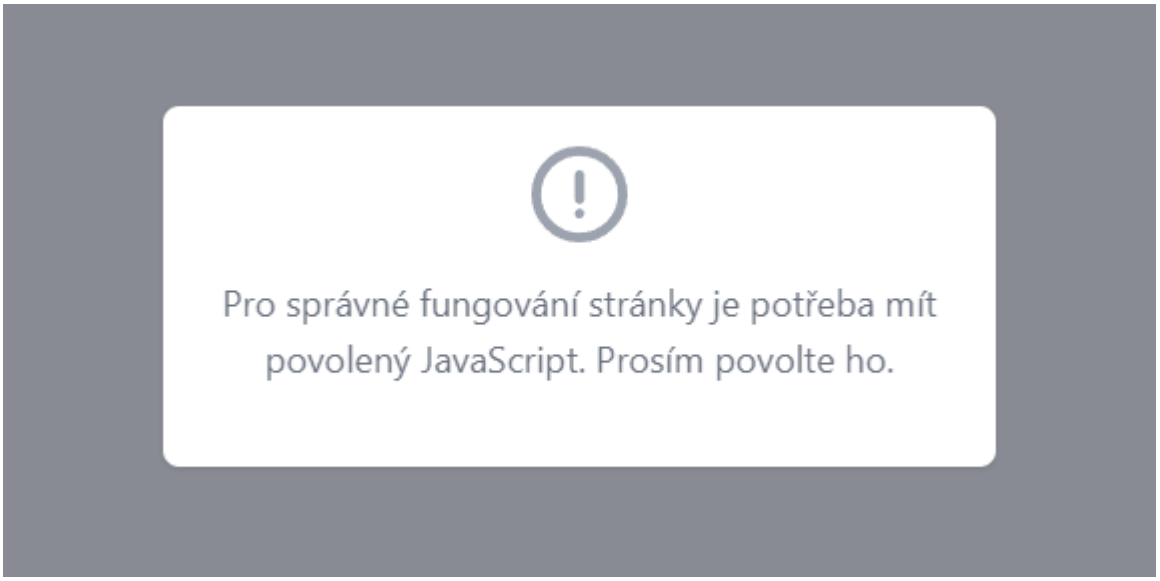
```
1  CREATE INDEX idx_service_client_id ON service(client_id);
2  CREATE INDEX idx_service_address_id ON service(address_id);
3  CREATE INDEX idx_service_status_id ON service(status_id);
4  CREATE INDEX idx_service_network_id ON service(network_id);
5  CREATE INDEX idx_service_nat_id ON service(nat_id);
6  CREATE INDEX idx_service_shaping_id ON service(shaping_id);
7  CREATE INDEX idx_service_created_by ON service(created_by);
8  CREATE INDEX idx_service_updated_by ON service(updated_by);
```

Obrázek 20 – Přidané indexy tabulky *service*

6.2 Struktura kódu aplikace

Aplikace je psána výhradně v jazyce PHP (konkrétně OOP). V PHP verze 8.2 je napsán celý back-end aplikace, tím je myšlena komunikace s databází, práce s daty, validace vstupních dat a více. Dále jsou využity jazyky HTML, CSS a JavaScript, které se výhradně používají pro grafické uživatelské prostředí (GUI – Graphical User Interface) aplikace. Celý tento kód se nachází ve složce společně s touto prací. Ve většině souborů se také nachází komentáře, které ve zkratce vysvětlují jednotlivé komponenty souborů.

Kvůli tomu, že funkčnost aplikace je závislá na funkcionalitě JavaScriptu, musí být vždy v prohlížeči zapnutý. Pokud tomu tak není, aplikace uživateli nedovolí se v aplikaci pohybovat a oznámí mu to pomocí modálu (Obrázek 21).



Obrázek 21 – Noscript modál

Níže bude představena základní struktura kódu aplikace a bude nastíněno, co obsahují jednotlivé soubory a složky. Některé části kódu budou představeny více dopodrobna až v dalších kapitolách.

6.2.1 Soubory a složky

Jednotlivé objekty jako klienti, služby a sítě mají své složky, ve kterých se poté uživatel aplikace pohybuje pomocí odkazů na stránce (např. v menu). Každá tato složka obsahuje soubory *index.php*, *add.php*, *update.php*, *delete.php* (v případě, že daný objekt lze smazat) a složku input, kde se nachází soubory *add-input.inc.php* a *update-input.inc.php*, které definují vstupy do objektů (Obrázek 23). Těmito objekty jsou nejčastěji myšleny objekty pro generaci HTML struktury pro danou stránku, či objekt pro zpracování vstupu z formuláře. Tyto definice umožňují rychle a efektivně měnit definice jednotlivých vstupů a pravidel, které musí daný vstup splňovat (Obrázek 22).

```
|   add.php
|   index.php
|   update.php
|
\---input
        add-input.inc.php
        update-input.inc.php
```

Obrázek 23 – Složka *clients*

```php
<?php

//Definice vstupů
$inputs = array(
    0 => array(
        'name' => 'public_ip4',
        'label' => 'Veřejná IPv4',
        'type' => 'text',
        'addons' => 'required placeholder="např. 185.249.114.93"',
        'location' => 'public_ip4',
        'form_group' => 1
    ),
    1 => array(
        'name' => 'note',
        'label' => 'Poznámka',
        'type' => 'textarea',
        'location' => 'note',
        'form_group' => 2
    ),
);

$update = array('nat');

$rules = array(
    0 => array(
            'ipAddress' => array(
                'type' => 'ip',
                'delegation' => 'public'
            ),
        'unique' => null
    ),
    1 => array(
        'max' => 100
    )
);

$order = array('nat');
```

Obrázek 22 – *add-input.inc.php* pro NAT

Výše zmíněné soubory jako *update.php, add.php* apod. obsahují základní definici HTML struktury pro daný soubor (včetně přidání javascriptových souborů a knihoven v hlavičce HTML). Dále také obsahují instance různých objektů, které slouží pro definici dalších HTML komponent, či pro inicializaci backendových funkcionalit.

Definice jednotlivých tříd/objektů, ve kterých se nachází všechny funkce a procedury (v rámci OOP nazývané také jako metody). Ty tvoří celou aplikaci a všechny se nachází ve složce *Classes*. Zde jsou soubory dále rozděleny do složek dle jejich povahy. Konkrétně do složek *Db, View, Controller*, tam jsou poté soubory rozděleny do dalších složek. Jednotlivé soubory jsou vždy pojmenovány stejně jako třída/objekt, který obsahují.

Db

Složka *Db* obsahuje soubory, které řeší konektivitu s databází, sestavení SQL dotazů pro databázi a předsestavené dotazy např. pro získání cizích klíčů tabulky, provedení vkladu dat do databáze či vytvoření samotné databáze (Obrázek 24).

```
+---Db
|       DatabaseConn.php
|       DBController.php
|       QueryBuilder.php
|
```

Obrázek 24 - Složka *Db*

View

Složka *View* obsahuje soubory pro generaci HTML obsahu. Ta je rozdělena dále do složek *Base* a *Component*.

Ve složce *Base* se nachází soubory pro generaci základních komponent jako je například menu nebo no-script modál. Pak také metody pro kombinaci jednotlivých základních komponent a funkcí, které se poté používají v předem zmiňovaných souborech jako například *update.php* (Obrázek 25).

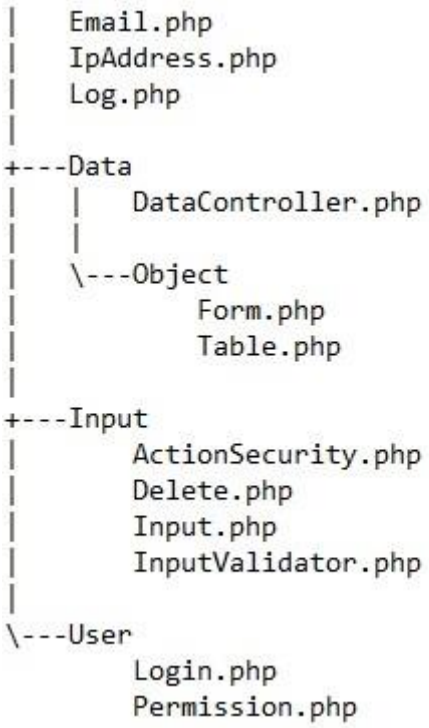
```
\---View
    +---Base
    |       BaseBodyView.php
    |       PageView.php
    |
    \---Component
            EmailStructure.php
            FormView.php
            TableView.php
```

Obrázek 25 – Složka *View*

Složka *Component* obsahuje soubory, které definují vzhled pro formuláře, přehledy (tabulky) a základní strukturu HTML emailu. Jednotlivé metody v objektu pro generaci generují vždy pouze část dané HTML struktury jako např. generace filtrů pro daný přehled, generace stránkování pro daný přehled, vygenerování vstupu pro formulář a tak podobně.

Controller

Složka *Controller* obsahuje všechny zbylé soubory s třídami/objekty a nachází se v ní nejvíce kódu. Obsahuje soubory, které slouží pro zpracovávání dat a procesů aplikace. Složka obsahuje objekty pro kontrolu logování, emailů, ip adres, session a dále složky *Data*, *Input* a *User* (Obrázek 26).



Obrázek 26 – Složka *Controller*

Složka *Data* obsahuje soubory, které zásadně pracují s daty aplikace. Nacházejí se zde soubory, které přímo definují metody pro práci s konkrétními daty v rámci zobrazení (tedy objekty jednotlivě pro formulář a tabulku). Dále je zde soubor *DataController*, který dědí všechny procesy z funkcí obsažených ve složce *Db*, zpracovává a vrací data pro zobrazení.

Další složkou je *Input*, která obsahuje soubory pro zpracování uživatelského vstupu. Obsahuje soubory, které zpracují a validují uživatelský vstup a sestaví z něj data použitelná pro bezpečný vklad do databáze. Dále je zde také soubor *ActionSecurity*, který obsahuje metody, které zajišťují bezpečnost v rámci akcí v aplikaci.

A jako poslední složka *User* obsahuje soubory, které se starají o informace o přihlášeném uživateli. Obsahují tedy proces přihlášení, odhlášení a práci s oprávněními.

Ostatní

Dále se ve struktuře nachází složky *login*, *data*, *log* a *assets*. Složka *login*, jak už je podle názvu zřejmé, obsahuje soubory pro zobrazení přihlašovací stránky a odhlášení. Složka *data* obsahuje PHP soubory, ke kterým se přistupuje pomocí javascriptového kódu v rámci responzivního webu (AJAX), takže například získání aktuálně vyplněného stavu ve formulářích pro službu. Složka *log* obsahuje předvytvořené soubory, do kterých aplikace loguje a složka *assets* obsahuje všechny potřebné CSS a JavaScript soubory.

6.2.2 Konfigurační soubor

Konfigurační soubor (*config.ini*) je důležitou součástí aplikace (Obrázek 27). Nachází se v základní složce aplikace a pomocí jeho definic se určují některé funkcionality v aplikaci. Nacházejí se v něm přesně 3 kategorie: *application*, *database* a *email*. Všechny zobrazené hodnoty v konfiguračním souboru musí být vyplněné validními hodnotami, tak jako v příkladu (demu). Pokud tento soubor nebude validně vyplněn, tak aplikace nebude fungovat. Tento soubor i jiné takto podstatné soubory jsou nepřístupné z prohlížeče, díky souborům *.htaccess*.

První kategorie *application* je tvořena polem *root_dir*, které obsahuje celou cestu do aplikace v rámci systému, a poté 3 boolean pole, které určují, zda se mají v dané oblasti tvořit logy. Kategorie *database* definuje všechna data potřebná pro připojení do databáze a kategorie *email* data potřebná pro automatickou komunikaci s klientem v rámci emailu.

```
[application]
root_dir = "C:/xampp/htdocs/oopv2/maturita_project/"
db_log = true
login_log = true
email_log = true

[database]
db_name = "isp"
db_user = "commander_app"
db_password = "%BGqRJXJ%zVQ"
db_host = "localhost"

[email]
client_notification = false
app_email = "pokorvo20@sps-prosek.cz"
app_email_password = "xtxzjzbaokjvgdyl"
name = "Test s.r.o"
#host = ""
```

Obrázek 27 – Soubor *config.ini*

Data z konfiguračního souboru jsou použita v souboru *constant.inc.php*, který se poté využívá v souborech, kde jsou potřeba data z konfiguračního souboru (Obrázek 28). Ty se ukládají do konstanty s názvem *CONFIG*, což je v podstatě Array s definovanými pravidly.

```php
<?php

$config = parse_ini_file('config.ini', true, INI_SCANNER_TYPED);
define("CONFIG", $config);
```

Obrázek 28 – Soubor *constant.inc.php*

6.3 Použité nástroje

V této kapitole budou představeny použité nástroje v rámci projektu. Pro jednotlivé nástroje je vždy jednoduše popsáno, o jaký nástroj se jedná, a poté také jeho využití v rámci projektu.

6.3.1 Composer

Composer je jednoduchý nástroj pro správu závislostí v rámci PHP. Usnadňuje distribuci knihoven a balíčků pro PHP, které jsou vnímány jako jednotlivé aplikační komponenty. Díky composeru se prostředí vývoje v PHP posunulo výše mezi jazyky s možností modernějšího stylu vývoje. Tento nástroj funguje tak, že pokud kód například přenášíte mezi zařízeními, tak nepřenášíte celý kód daných balíčků, ale pouze definiční soubor composeru. V tomto JSON souboru (*composer.json*) (Obrázek 29) je definováno, jaké verze a balíčky se mají stáhnout, pak už stačí jen zadat příkaz pro stažení (Obrázek 30) nebo přidání požadovaného balíčku s jeho verzí do souboru (Obrázek 29). Kompletní instalace composeru (pro Windows a Linux) je popsána přímo na jeho webových stránkách composeru viz. https://getcomposer.org/doc/00-intro.md.

Obrázek 29 – Soubor *composer.json*

```
vojtech@postgre:/var/www/vojtapoky/maturita_project$ composer require phpmailer/phpmailer
./composer.json has been updated
Running composer update phpmailer/phpmailer
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking phpmailer/phpmailer (v6.9.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing phpmailer/phpmailer (v6.9.1): Extracting archive
6 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating autoload files
2 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^6.9 for phpmailer/phpmailer
vojtech@postgre:/var/www/vojtapoky/maturita_project$
```

Obrázek 30 – Instalace balíčku phpmailer

V praktické části projektu je composer využitý pro přehlednou a jednoduchou správu balíčků PHPmailer a Monolog, které jsou níže představeny podrobněji. Zároveň se využívá tzv. autoloader composeru, který automaticky načítá soubory s definovanými objekty do souborů, kde jsou potřeba, i pro vlastní soubory (objekty) vytvořené v rámci aplikace. V aplikaci a v demu se využívá přesně verze composeru 2.6.6. (45) (46)

6.3.2 PHPMailer

PHPMailer je nástroj, který umožňuje pokročilé funkce pro zasílání emailů v rámci PHP. Tento balíček pomáhá do základního odesílání emailů přidat funkce jako šifrování komunikace, autentizace, zasílání HTML emailů a zasílání příloh v rámci emailu. PHP sice obsahuje funkce pro základní odesílání emailů, ta ale neumožňuje kromě samotného odesílání skoro nic. Instalace balíčku je poté jednoduchá pomocí již v práci zmiňovaného composeru.

V projektu je využita knihovna pro její jednoduchost a zmiňované možnosti, a to hlavně možnosti použití HTML struktury a šifrování. Dále lze jednoduše definovat odesílání emailu a jeho vlastnosti a jde využít například emailového serveru Googlu a svého účtu pro odesílání. To, jak využít Gmail pro odesílání emailů skrze PHPmailer, je popsáno v tomto tutoriálu https://mailtrap.io/blog/phpmailer-gmail/. V projektu je pak použitá konkrétně verze PHPMailer 6.9. (47)

PHPmailer je využitý ve funkci pro odesílání emailů s názvem *sendEmail*. Ve funkci se používají údaje definované v konfiguračním souboru, které jsou uložené v konstantě *CONFIG*, a dále také vstupní atributy pro danou funkci. Ty tvoří HTML strukturu emailu, která je definována pro daný email, příjemce, předmět a také alternativní tělo emailu, které se využije v případě, že HTML strukturu u daného příjemce nelze zobrazit. Na začátku funkce se nachází kontrola, zda je odesílání emailů v rámci aplikace vůbec povolené a dále instance objektů *PHPMailer* a *Log*. Poté se nastavují všechny parametry pro daný email a odesílá se. Mezi nastavované údaje patří např. šifrování, pro které je konkrétně použitý protokol TLS a změna portu na 587. Pokud nastane chyba ve chvíli, kdy se nastavuje a odesílá email, tak se email neodešle a chyba se zapíše do logu. Chyby a úspěšně odeslané emaily se zapisují do logu pouze ve chvíli, kdy je v konfiguračním souboru logování povoleno. Logování je konkrétněji popsané v kapitole týkající se nástroje Monolog. Níže se nachází ukázka části kódu popisované funkce, ve které se nachází definice samotného odesílání emailu pomocí PHPMailer (Obrázek 31).

```php
$mail->isSMTP();
$mail->Host = CONFIG['email']['app_email']['host'] ?? 'smtp.gmail.com';
$mail->SMTPAuth = true;
$mail->Username = CONFIG['email']['app_email'];
$mail->Password = CONFIG['email']['app_email_password'];
$mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
$mail->Port = 587;
$mail->CharSet = "UTF-8";
$mail->Encoding = 'base64';

//Odesílatel a příjemce

$mail->setFrom(CONFIG['email']['app_email'], CONFIG['email']['name'] ?? 'ISP Commander');
$mail->addAddress($input['receiver']);
$mail->addReplyTo(CONFIG['email']['app_email']);
```

Obrázek 31 – Část funkce *sendEmail*

6.3.3 Monolog

Monolog je nástroj, který umožňuje implementaci logování v rámci PHP. Monolog umožňuje implementovat jednoduché i pokročilé techniky logování v PHP aplikacích. Nabízí různé úrovně logování a techniky jako logování do textových souborů, databáze a podobně. V aplikaci a v demu je využita verze 3.5.

Monolog je použit v souboru *Log*, ve kterém jsou definované funkce pro logování různých událostí do textových souborů. Tyto funkce jsou použity různě napříč aplikací, pro účely diagnostiky a analýzy.

Níže je příklad použití nástroje ve funkci *databaseAction* (Obrázek 32). Ta nejdříve zkontroluje, zda je logování povoleno v konfiguračním souboru. Pokud ano, vytvoří instanci objektu *StreamHandler*, který je součástí monolog a použije definovanou cestu pro logovací textový soubor. Poté sestaví potřebná data do datové struktury a přidá nový řádek do souboru.

```php
public function databaseAction($actionSQL, $query, $queryData)
{

    if (CONFIG['application']['db_log'] !== true) {
        return;
    }

    $handler = new StreamHandler($this->dbLogFile);
    $this->databaseLogger->pushHandler($handler);

    $logData = array(
        'username' => Session::get(array('user_logged', 'username'), 1),
        'query' => $query,
        'query_data' => $queryData
    );

    $this->databaseLogger->info('Byl proveden ' . $actionSQL . ' v databázi', $logData);
}
```

Obrázek 32 – Funkce *databaseAction* v *Log*

6.3.4 Tailwind CSS

Tailwind CSS je nástroj pomocí kterého lze rychle a snadně stylovat webové stránky. Jedná se o CSS framework pro vlastní vytváření uživatelského rozhraní, které umožňuje inline stylování pomocí definovaných HTML class (tříd) přímo v HTML. Tailwind poté skenuje všechny HTML soubory a jejich komponenty, dle kterých vygeneruje statický CSS soubor, který obsahuje definice všech použitých class. V dnešní době je nástroj velmi populární a využívá se místo vytváření samotných CSS souborů, díky jeho flexibilitě, rychlosti, spolehlivosti a přehlednosti. (48) (49)

V projektu je nástroj použit pro stylování všech komponent v rámci frameworku postaveného na Tailwind CSS zvaného Flowbite. Pro generaci CSS souboru bylo využito nasazení Node.js serveru na lokální stroj, skrze který poté Tailwind generoval daný CSS soubor, který je přidaný do všech HTML souborů. Návrhy a všechny styly byly nejdříve vytvořeny jako statické HTML stránky a jejich HTML kód byl pak následně rozdělen do různých PHP souborů pro generaci stránky. Ve finálním projektu se nachází pouze finální CSS soubor.

6.3.5 Flowbite

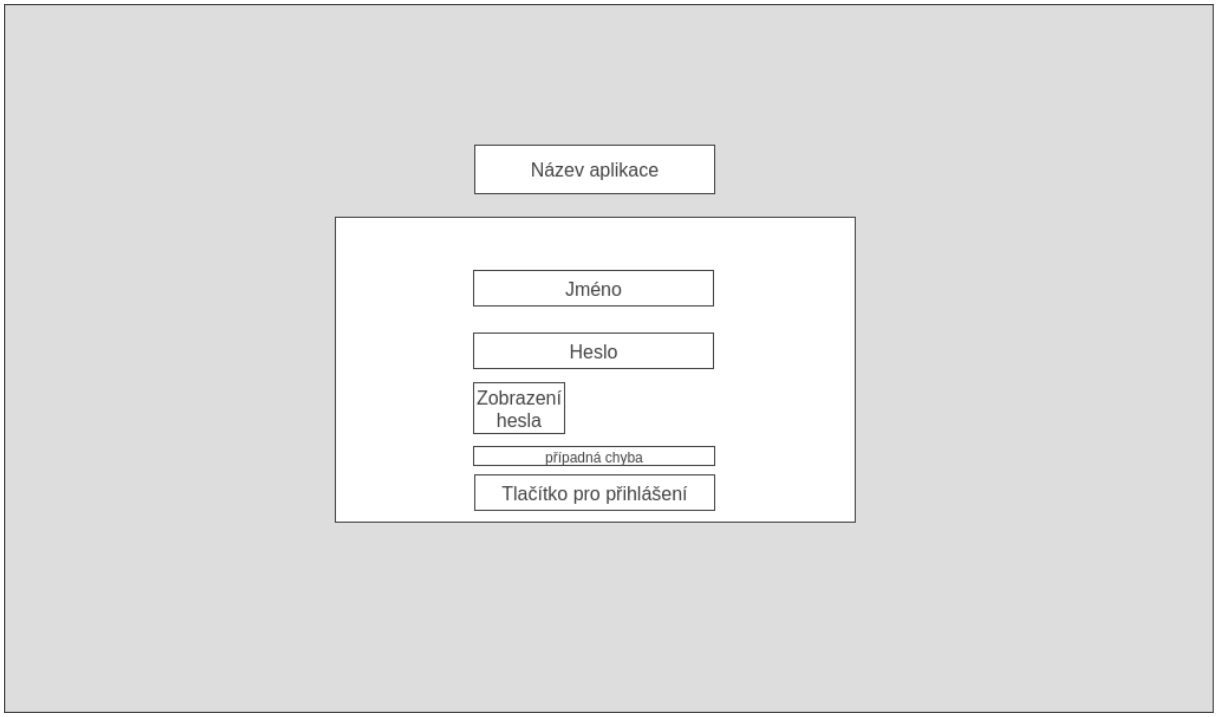
Flowbite je open-source knihovna HTML, JavaScript a dalších komponent, které využívají framework Tailwind CSS. Obsahuje všechny běžné komponenty, které webové stránky používají, jako jsou tlačítka, menu, lišty, modály a také další interaktivní prvky. Každý prvek je vytvořen pomocí Tailwind CSS tříd a základního JavaScriptu. (50)

V projektu je většina komponenta právě z této knihovny. Následně je většina komponent dodatečně upravena pomocí přidání, nebo smazání Tailwind CSS HTML class (tříd). JavaScript knihovna je importována pomocí CDN odkazu, který se nachází v hlavičce HTML souborů (Obrázek 33).

Obrázek 33 – CDN Flowbite

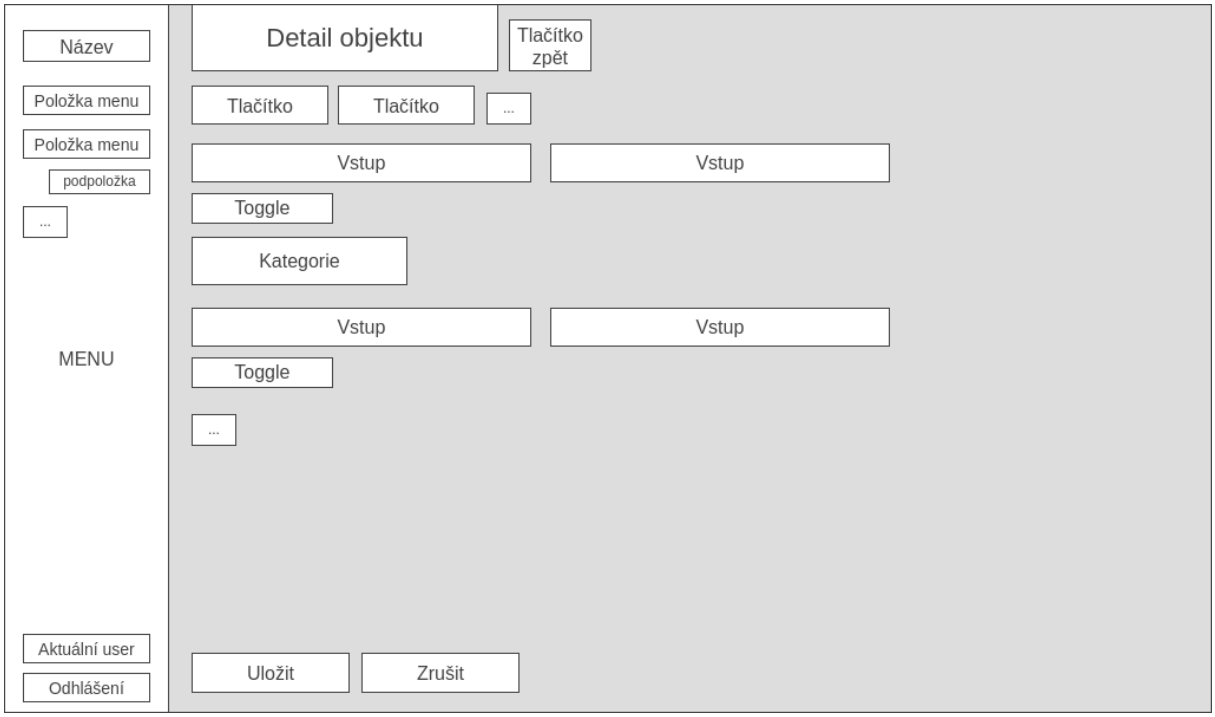
6.4 Návrh GUI

Pro návrh GUI (Graphical User Interface) bylo pro aplikaci vytvořeno několik základních wireframů, dle kterých se poté sestavovala samotná struktura vzhledu jednotlivých komponent aplikace. Pomocí výše zmíněných nástrojů Tailwind CSS a Flowbite se dle wireframových návrhů použily komponenty a sestavil se layout celé aplikace. Vždy se vybraly komponenty z knihovny Flowbite, které se na dané místo v návrhu nejvíce hodily a popřípadě se komponenty ještě lehce poupravily. Díky použití komponent z knihovny Flowbite je aplikace také responzivní a přináší některé jednoduché animace. Níže jsou zobrazeny základní wireframy, dle kterých se jednotlivé stránky tvořily (Obrázek 34) (Obrázek 35) (Obrázek 36).



Obrázek 34 – Wireframe přihlášení

Obrázek 35 – Wireframe přehled



Obrázek 36 – Wireframe formulář

6.5 Nasazení aplikace

Pro nasazení aplikace je potřeba nejdříve nasadit na server tzv. LAPP (Linux Apache Postgres PHP) stack. To znamená, že na linuxovou serverovou distribuci se nasadí webový server s PHP a PostgreSQL databází. Na tomto serveru poté bude nasazena samotná aplikace. Konkrétní ukázka nasazení aplikace v této kapitole bude mít stejné parametry jako nasazení dema. Přístupové údaje k demu se nacházejí v příloze.

6.5.1 Příprava serveru

Demo běží na linuxové distribuci Debian verze 11. Aplikace byla nasazena na této distribuci a jinak byla vyvíjena pomocí lokálního webového serveru na Windows pomocí nástroje Xampp (viz. https://www.apachefriends.org/) a lokálně staženého PostgreSQL serveru.

Jako první je potřeba nasadit webový (HTTP) server. Je potřeba využít přímo Apache, protože některé funkce aplikace jsou na tom závislé, a to například použití *.htaccess* souborů pro zabezpečení některých souborů (viz. Apache, Debian 11 tutoriál). Dále je potřeba nainstalovat PHP, aplikace je napsána a testována ve verzi 8.2 (viz. PHP, Debian 11 tutoriál). Použití *.htaccess* souborů musí být implicitně povoleno jinak také nebudou fungovat. Povolit a

nastavit to lze přímo v konfiguračním souboru Apache a v konfiguračním souboru pro konkrétní doménu (viz. .htaccess tutoriál). Poté je tu ještě zmíněný nástroj composer, bez kterého nelze provést iniciální nastavení, takže se také musí nainstalovat.

Dále je také důležité pro danou doménu získat SSL/TLS certifikát, aby aplikace běžela na HTTPS. Existuje pro to jednoduchý nástroj Certbot, který umožňuje získat certifikáty z certifikační autority Let's Encrypt (viz. Certbot, Debain 11 tutoriál). Tento nástroj byl využit také při nasazení dema aplikace.

Poslední instalace, která je potřeba, je PostgreSQL (konkrétně verze 15). V *php.ini* je také nutné odkomentovat rozšíření týkající se pgsql, aby mohla komunikace PHP s databází fungovat (viz. PostgreSQL, Debian tutoriál).

6.5.2 Iniciální nastavení

Jakmile máte připravený server, tak dále lze přidat zdrojový kód samotné aplikace. Kód se nachází ve složce společně s touto prací, ten je možno vzít a převést ho na server například pomocí FTP (File Transfer Protocol – protokol využívaný pro přenos souborů mezi zařízeními v síti), nebo ideálně jednoduše pomocí nástroje git, který je potřeba doinstalovat na server. Tento způsob je také použitý v ukázce, která je níže. V ní je předveden postup pro iniciální konfiguraci aplikace (ve příkazech se musí nahradit cesty do jednotlivých složek).

Nejdříve se naklonuje zdrojový kód z privátního repozitáře nacházejícího se na GitHubu. Pro ověření se musí v promptu použít uživatelské jméno a PAT (Personal Access Token), protože je repozitář privátní. Oba tyto údaje jsou zahrnuty v komentářích níže. Poté jsou upravena oprávnění pro složky, aby se mohli provést potřebné záležitosti v rámci composer příkazů. Poté je nutné v konfiguračním souboru (Obrázek 27) upravit položku *root_dir*, upravit údaje pro připojení do databáze, které se poté využijí pro vytvoření uživatele, skrze kterého bude aplikace přistupovat do databáze a popřípadě upravit další položky (konfigurační soubor je popsán výše v práci v kapitole věnované struktuře kódu aplikace). Dále se musí odebrat a znovu přidat potřebné composer balíčky a přegenerovat soubor autoload. Po provedení těchto akcí se vlastník složky změní zpět na uživatele *www-data*.

V další pasáži příkazů je nutné se přihlásit na uživatele *postgres*, který má přístup do základní stejnojmenné databáze. Zde je nutné předvytvořit databázi a následně PostgreSQL uživatele, který se bude používat pro připojení aplikace do databáze, kterého jsme už definovali

v konfiguračním souboru. V aplikaci je použit uživatel se jménem *commander_app*. Následně se v prohlížeči otevře (v prohlížeči) soubor *init.php*, který se nachází v základní složce zdrojového kódu aplikace. Tento soubor automaticky vytvoří celou strukturu databáze. Po dokončení načítání daného souboru je nutné definovat vlastnictví databáze a odebrat danému uživateli možnost vytvářet další databáze. A jako poslední se odstraní samotný soubor *init.php*, či se přemístí mimo dostupný webový adresář.

```
# Získání zdrojového kódu
# VojtaPokorny
# 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
sudo git clone https://github.com/VojtaPokorny/maturita_project.git
/var/www/vojtapoky/maturita_project

sudo chmod -R 755 /var/www/vojtapoky/maturita_project
sudo chown -R $USER:$USER /var/www/vojtapoky/maturita_project/

# Zde je potřeba v konfiguračním souboru upravit root_dir
sudo mcedit /var/www/vojtapoky/maturita_project/config.ini

# Přenačtení composer balíčků a autoload souboru
cd maturita_project/
composer remove monolog/monolog
composer remove phpmailer/phpmailer
composer require monolog/monolog
composer require phpmailer/phpmailer
composer dump-autoload
sudo chown -R www-data:www-data /var/www/vojtapoky/maturita_project/

# Následující potřeba provádět pod uživatelem, který má přístup do databáze (postgres)
psql -d postgres
```

```
CREATE DATABASE isp;
CREATE USER commander_app WITH PASSWORD '%BGqRJXJ%zVQ';
GRANT ALL PRIVILEGES ON DATABASE isp TO commander_app;
ALTER DATABASE isp OWNER TO commander_app;
ALTER USER commander_app CREATEDB;

# ! DÁLE JE NUTNÉ SPUSTIT V PROHLÍŽEČI SOUBOR init.php, který se nachází v základní
složce aplikace !

ALTER USER commander_app NOCREATEDB;
GRANT ALL PRIVILEGES ON DATABASE isp TO commander_app;
ALTER DATABASE isp OWNER TO commander_app;
\q

# Odebrání souboru init.php
sudo rm -f /var/www/vojtapoky/maturita_project/init.php
```

6.6 Funkcionalita a příklady kódu

V této kapitole budou představeny některé konkrétní funkce a objekty použité v rámci kódu aplikace. V rámci každé konkrétní funkce bude popsána její funkcionalita, a kde ve struktuře je použita. Pro větší funkce není v textu přímo přiložen jejich kód, ale např. pouze ukázka. Celý kód je v případě potřeby ve složce se zdrojovým kódem, který se nachází ve stejné složce jako tento text. To, kde se zrovna daná funkce nachází, lze dohledat výše v kapitole (Soubory a složky). Co se poté týče popisu konkrétních objektů/tříd, ty budou popsány více obecně.

6.6.1 Přihlášení a oprávnění

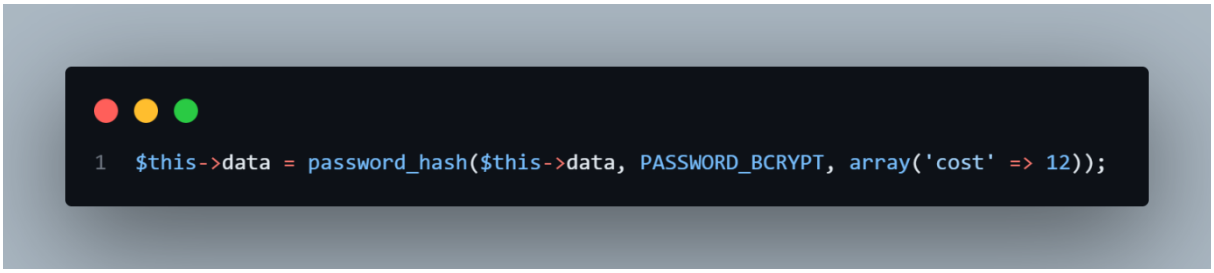
Záležitosti týkající se přihlašování do aplikace řeší objekt *Login*. Řeší konkrétně přihlášení, odhlášení a kontrolu, zda je uživatel přihlášený v rámci stránky. To, zda je uživatel

přihlášený, kontroluje pomocí parametru uloženého v SESSION, který se smaže ve chvíli, kdy se uživatel odhlásí. SESSION je globální proměnná, do které se dočasně ukládají data na straně serveru.

Samotné přihlašování probíhá pomocí zavolání funkce *action*. V této funkci se vytvoří instance *InputValidator* (objektu určeného pro kontrolu vstupů). Pomocí validátoru se data zbaví všech možných tagů a jiných nevalidních částí vstupu pomocí funkce *sanitization* (Obrázek 37). Poté zavolá funkci *loginCheck*, která zkontroluje dané vstupy, vrátí boolean, který říká, zda byl uživatel úspěšně přihlášen a případně vrátí chybovou hlášku. Konkrétně zkontroluje, zda dané přihlašovací jméno vůbec existuje, zda je daný účet aktivní, a poté teprve jestli bylo zadáno správné heslo. Heslo je hashované pomocí PHP funkce *password_hash*, která využívá algoritmus bcrypt (Obrázek 38). Pokud se uživatel úspěšně přihlásí, tak se uloží informace o něm (včetně oprávnění) do SESSION.

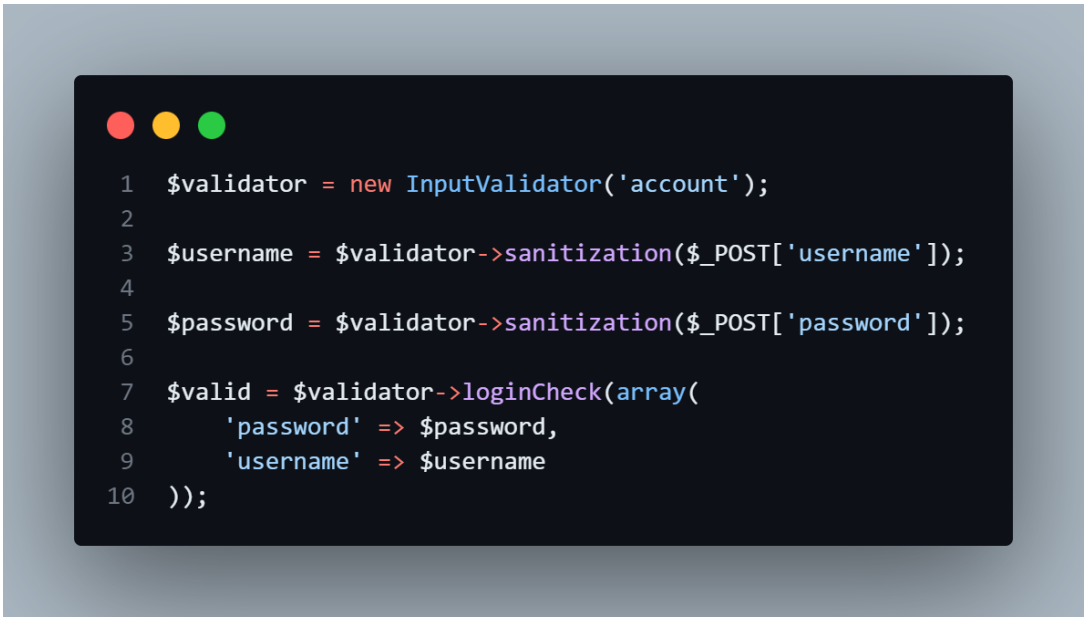
```php
/**
 * Sanitizace vstupních dat uživatele
 *
 * @param mixed $data vstup, který se má sanitizovat
 */
public function sanitization($data)
{

    $data = trim($data);
    $data = str_replace('?>', '', $data);
    $data = strip_tags($data);
    $data = htmlspecialchars($data, ENT_QUOTES, 'UTF-8');

    return $data;
}
```

Obrázek 37 – Funkce *sanitization*



```php
$this->data = password_hash($this->data, PASSWORD_BCRYPT, array('cost' => 12));
```

Obrázek 38 – Funkce *password_hash*



```php
$validator = new InputValidator('account');

$username = $validator->sanitization($_POST['username']);

$password = $validator->sanitization($_POST['password']);

$valid = $validator->loginCheck(array(
    'password' => $password,
    'username' => $username
));
```

Obrázek 39 - Část funkce *action* v *Login*

Oprávnění řeší v aplikaci objekt *Permission*. Nejčastěji se pak používá jeho funkce *check*, které se při vstupu specifikuje, jaké musí mít přihlášený uživatel oprávnění. Tyto oprávnění jsou pro uživatele uloženy při přihlašování do SESSION. Pokud požadované oprávnění nemá je vrácen zpět na přehled. Uživatelům se také nezobrazují např. tlačítka na smazání, pokud mazat nemohou.

6.6.2 Připojení k databázi a dotazy

Tyto záležitosti jsou řešeny v rámci už zmíněné složky *Db*. Ta řeší konektivitu s databází, sestavení SQL dotazů pro databázi a předsestavené dotazy. Obsahuje konkrétně definice objektů *DatabaseConn*, *DBController* a *QueryBuilder*, které jsou níže jednotlivě popsány.

Objekt *DatabaseConn* slouží pro spojení s databází pomocí PDO (PHP Data Object). PDO je ve zkratce rozšíření v PHP, které nabízí bezpečnější a rozšířenější možnosti pro

komunikaci s různými databázovými systémy (51). Samotné spojení je navázané ve funkci *_construct* (Obrázek 40), která se automaticky zavolá ve chvíli, kdy se vytváří nová instance tohoto objektu. Obsahuje základní definice spojení získané z konstanty CONFIG, a dále nastavení pro dané spojení.

```php
/**
 * Incializace připojení do databáze
 *
 * Připojení je pouze na POSTRGRESQL databázi
 * Proměnné jsou určeny v konfiguračním souboru
 */
public function __construct()
{

    $options = [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES => false,
    ];

    $def = "pgsql:host=" . CONFIG['database']['db_host'] . ";dbname=" . CONFIG['database']['db_name'];

    try {

        $conn = new PDO($def, CONFIG['database']['db_user'], CONFIG['database']['db_password'], $options);
        $conn->exec("
            SET CLIENT_ENCODING TO 'utf8';
            SET TIMEZONE TO 'Europe/Prague';
        ");
        $this->conn = $conn;
    } catch (PDOException $e) {

        $log = new Log();
        $log->databaseError("Connection failed: " . $e->getMessage(), 'Connection', $def);
    }
}
```

Obrázek 40 - Funkce *_construct* v *DatabaseConn*

Dalším objektem je *QueryBuilder*, pomocí kterého se sestavují SQL dotazy pro získání dat do tabulek a formulářů. Ta obsahuje jednotlivé funkce, pomocí kterých se definují jednotlivé komponenty dotazu, jako jsou podmínky (Obrázek 41), selekce a limitace. Jakmile se tyto komponenty všechny nadefinují, tak se zavolá funkce *build* (Obrázek 42). Ta samotný dotaz sestaví, a poté ho uloží do své proměnné.

```php
/**
 * Přidání podmínky do query
 *
 * @param string $conditionLeftSide sloupec na který se vztahuje podmínka
 * @param string $compare operátor použitý v podmínce
 * @param string $conditionRightSide podmínka, za operátorem
 * @param string $operator AND/OR/...
 */
public function where($conditionLeftSide, $compare, $conditionRightSide, $operator = 'WHERE')
{

    //$this->query .= ' ' . $operator . ' ' . $conditionLeftSide . ' ' . $compare . ' ' . $conditionRightSide;

    $this->whereConditions[] = array(
        'conditionLeftSide' => $conditionLeftSide,
        'compare' => $compare,
        'conditionRightSide' => $conditionRightSide,
        'operator' => $operator
    );

    return $this;
}
```

Obrázek 41 – Funkce *where* v *QueryBuilder*

```php
$queryBuilder = new QueryBuilder('nat');

$queryBuilder
    ->select(array(
        'nat.id AS id', 'ac.username AS created_by', 'au.username AS updated_by',
        'nat.created_at', 'nat.updated_at',
        'nat.public_ip4', 'nat.note', 'service.id AS service_id', 'service.service_name'
    ))
    ->leftJoin('service', 'id', 'nat', 'service_id')
    ->leftJoin('account', 'id', 'nat', 'created_by', 'ac')
    ->leftJoin('account', 'id', 'nat', 'updated_by', 'au')
    ->groupBy('
        nat.id, ac.id, au.id, service.id
    ')
    ->orderBy('nat.id', 'ASC');
```

Obrázek 42 – Příklad použití *QueryBuilder*

Posledním objektem je *DBController*, ten obsahuje předem definované dotazy a skripty pro definici struktury databáze. Předem definovanými dotazy jsou myšleny dotazy pro vklad a mazání dat v databázi, dále získání cizích klíčů použitých v tabulce a tak dále. Dále se zde nachází funkce *executeQuery* (Obrázek 43), která slouží pro spuštění dotazu nacházejícím se v instanci obejtku *QueryBuilder*.

```php
/**
 * Provedení sestaveného dotazu (query)
 *
 * @param object $queryBuilder objekt QueryBuilder
 */
public function executeQuery($queryBuilder)
{

    $queryBuilder->build();

    $query = $queryBuilder->query;
    $prepare = $queryBuilder->prepare;

    $sqlProv = $this->conn->prepare($query);

    if (isset($prepare)) {
        $sqlProv->execute($prepare);
    } else {
        $sqlProv->execute();
    }

    $data = $sqlProv->fetchALL(PDO::FETCH_ASSOC);

    return $data;
}
```

Obrázek 43 - Funkce *executeQuery*

6.6.3 Uživatelské vstupy

Zaznamenávání uživatelských vstupů do databáze je jednou z nejdůležitějších funkcí aplikace. Pro tyto účely slouží výhradně objekty *Input*, *InputValidator* a další objekty, které zajišťují bezpečnost a podpůrnou validaci.

Nejdůležitějším objektem je *Input*. Ten řeší všechny uživatelské vstupy v rámci sestavení dat pro užití v jednotlivých dotazech, různá rozhodnutí o tom, jaké metody pro uložení se mají využít apod. Funkce a postupy v objektu jsou velmi obecné a jsou napsané tak, aby byly pro vývojáře škálovatelné pomocí modifikace definicí nacházejících se v souborech

add-input.inc.php a *update-input.inc.php*, které byly zmíněny výše v kapitole o struktuře kódu aplikace.

Objekt nejdříve vyřeší kontrolu validity vstupů. Jakmile vše zkontroluje, tak buďto vrátí chybu a předvyplní správně vyplněné části formuláře, nebo pokračuje dál pomocí volání dalších jeho funkcí. V těchto funkcích (*dataUpdate* a *dataInsert*) sestavuje jednotlivá vstupní data pro jednotlivé dotazy, které zapíšou výslednou úpravu ve formuláři do databáze. Dále se rozhoduje například o tom, jaký bude použit příkaz (UPDATE, INSERT nebo DELETE).

V objektu se nachází také hodně pomocných funkcí. Jedna z těchto funkcí je *replaceInputPlaceholder* (Obrázek 45), která nahrazuje definované pole, do kterého se má vyplnit daný vstup (Obrázek 44). To, kam přesně se má v datové struktuře vstup vyplnit, je označené pomocí placeholderu *{input}*. Dále existují také placeholdery pro potřebná ID ve datové struktuře, ty jsou značeny *{NAZEV_OBJEKTU}* (např. *{client}*). Nahrazovní těchto placeholderů samotnými identifikátory probíhá pomocí další funkce zvané *replaceIdPlaceholders*.

```php
array(
    'name' => 'primary_email',
    'label' => 'Primární E-mail',
    'type' => 'text', 'addons' => 'placeholder="example@gmail.com"',
    'location' => 'contacts/contact->type=Primární email',
    'form_group' => 'Kontakty',
    'nest' => array(
        'contact' => array(
            'contact' => '{input}',
            'client_id' => '{client}',
            'contact_type_id' => '1',
        )
    )
);
```

Obrázek 44 - Definice vstupu *Primární-Email* (formulář klient)

```php
/**
 * Nahrazení placeholderů, které jsou označené {input}
 * {object} musí být řešena až u samotného vkládání, protože nemusí být známá
 *
 * Dále tato metoda řeší přidání lokace v datové struktuře objektu do input dat pro action update
 *
 * @param array $data datová struktura ve které cheme nahradit {input}
 * @param mixed $inputValue čím se má nahradit placeholder
 * @param number $inputNum číslo daného vstupu (v definičním arrayi $input)
 */
private function replaceInputPlaceholder(&$data, $inputValue, $inpuNum)
{

    foreach ($data as $key => &$value) {

        if (is_array($value)) {

            $cacheArray = array();

            foreach ($value as $column => $columnValue) {

                $newKey = ':' . $column;

                //Kontrola, aby se případně nepřidával do databáze redundantí obsah
                //Pokud je pro daný {input} získané ID odstraní se z vkladové struktury dat

                if ($this->includeRepeat[$inpuNum] === true) {

                    $newValue = str_replace('{input}', $inputValue, $columnValue);

                    $cacheArray[$newKey] = !empty($newValue) ? $newValue : null;
                } elseif(!str_contains($columnValue, '{input}')) {

                    $newValue = str_replace('{input}', $inputValue, $columnValue);

                    $cacheArray[$newKey] = !empty($newValue) ? $newValue : null;
                }
            }

            $data[$key] = $cacheArray;
        }
    }
}
```

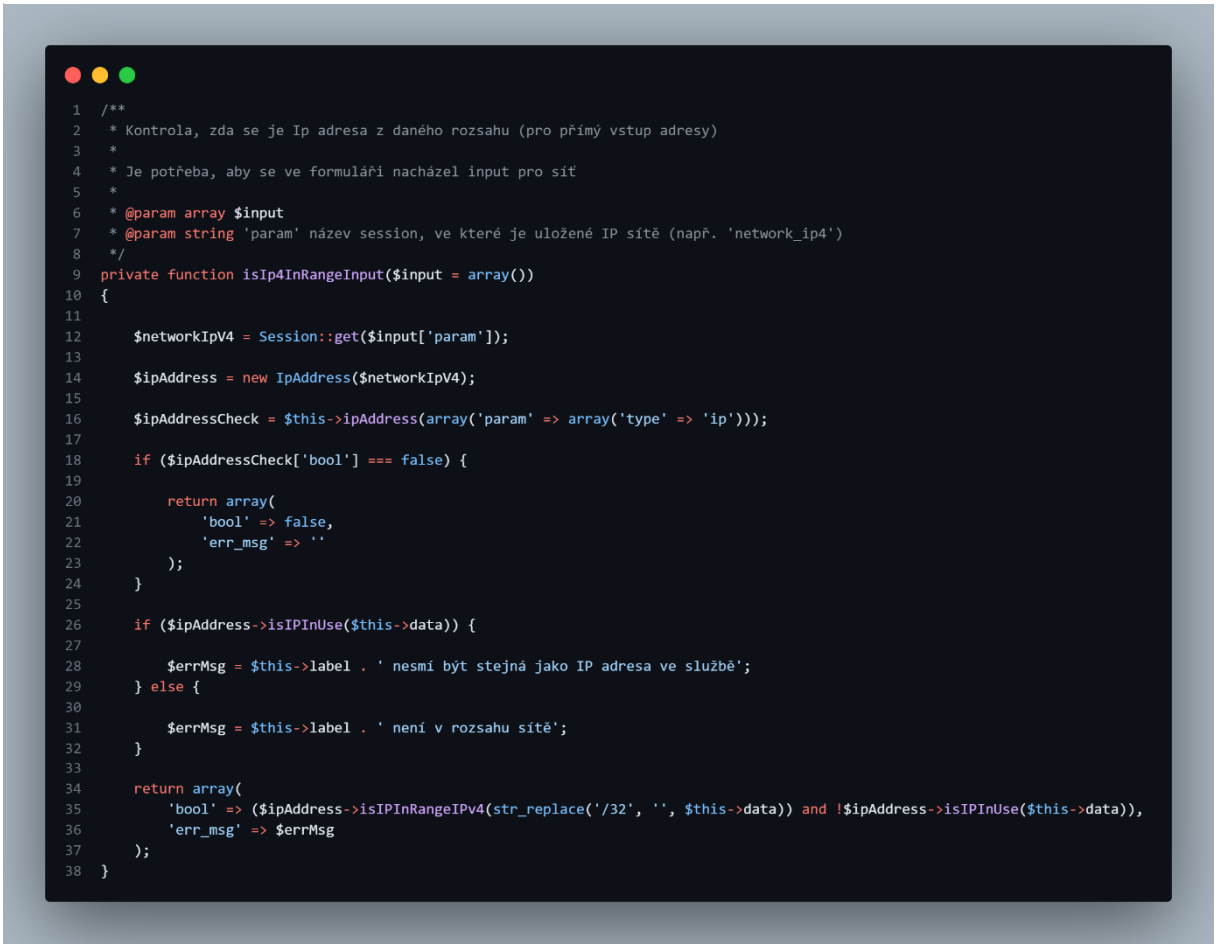
Obrázek 45 - Funkce *replaceInputPlaceholder*

Objekt *InputValidator* se využívá v rámci popisovaného objektu *Input*. Neobsahuje funkce pouze pro kontrolu vstupů do formuláře, ale také například kontrolu GET parametrů nacházejících se v URL nebo kontrolu pro přihlášení zmíněnou výše. Pro jednotlivé vstupy volá kontrolní funkce, které jsou opět definovány v souborech *add-input.inc.php* a *update-input.inc.php*. Jednotlivé kontrolní funkce obsahují jednoduchý, či složitější postup kontroly a vrací boolean, který určuje, zda vstup prošel danou kontrolou. Pokud danou kontrolou neprojde,

tak funkce zároveň vrací hlášku, která se zobrazí nahoře ve formuláři jako chyba. Níže se nachází konkrétní příklady těchto kontrolních funkcí. Konkrétně funkce *macAdress* (Obrázek 46) a *isIp4InRangeInput* (Obrázek 47). První z funkcí řeší, zda je vstup validní MAC adresa a druhá řeší, zda je vyplněná IP adresa v rozsahu vyplněné sítě.

```php
/**
 * Kontrola, zda se jedná o MAC adresu
 */
private function macAddress($input = array())
{

    return (array(
        'bool' => (filter_var($this->data, FILTER_VALIDATE_MAC) and str_contains($this->data, ':')),
        'err_msg' => $this->label . ' neodpovídá struktuře MAC adresy (MAC musí být ve formátu XX:XX:XX:XX:XX:XX)'
    ));
}
```

Obrázek 46 – Funkce *macAddress* v *InputValidator*



```php
/**
 * Kontrola, zda se je Ip adresa z daného rozsahu (pro přímý vstup adresy)
 *
 * Je potřeba, aby se ve formuláři nacházel input pro síť
 *
 * @param array $input
 * @param string 'param' název session, ve které je uložené IP sítě (např. 'network_ip4')
 */
private function isIp4InRangeInput($input = array())
{

    $networkIpV4 = Session::get($input['param']);

    $ipAddress = new IpAddress($networkIpV4);

    $ipAddressCheck = $this->ipAddress(array('param' => array('type' => 'ip')));

    if ($ipAddressCheck['bool'] === false) {

        return array(
            'bool' => false,
            'err_msg' => ''
        );
    }

    if ($ipAddress->isIPInUse($this->data)) {

        $errMsg = $this->label . ' nesmí být stejná jako IP adresa ve službě';
    } else {

        $errMsg = $this->label . ' není v rozsahu sítě';
    }

    return array(
        'bool' => ($ipAddress->isIPInRangeIPv4(str_replace('/32', '', $this->data)) and !$ipAddress->isIPInUse($this->data)),
        'err_msg' => $errMsg
    );
}
```

Obrázek 47 – Funkce *IsIp4InRangeInput* v *InputValidator*

Dále se v aplikaci řeší zabezpečení proti SQL injection, a to konkrétně v jednotlivých funkcí pomocí přípravy a dosazení vstupů, které nabízí PDO. Dále existuje objekt *ActionSecutrity*, který řeší původ daných HTTP dotazů, a to zda pocházejí přímo z aplikace. V rámci objektu se nacházejí funkce pro kontrolu metody, původu AJAX dotazu a generace s kontrolou CSRF (Cross-Site Request Forgery) tokenu. Níže se nachází příklady některých těchto funkcí (Obrázek 48) (Obrázek 49) (Obrázek 50).

CSRF token má zajistit ochranu proti stejně zvaným útokům, jejichž cílem je provést akci za uživatele, který tuto akci nezamýšlel, a to například ve chvíli, kdy bude mít uživatel otevřenou tuto aplikaci a zároveň web útočníka. Tomu lze zabránit generací tzv. CSRF tokenu, pomocí kterého si stránka ověří, zda požadovaná akce pochází z legitimního zdroje. Proto také v některých případech nelze mít aplikaci otevřenou ve více oknech a provádět některé akce nebo se vracet v aplikaci na formuláře pomocí šipek v prohlížeči. (52)

```php
/**
 * Kontrola, zda se jedná o metodu POST
 */
public static function isPost()
{

    if ($_SERVER['REQUEST_METHOD'] !== 'POST') {

        header('HTTP/1.1 405 Method Not Allowed');
        exit();
    }
}
```

Obrázek 48 – Funkce *isPost* v *ActionSecurity*

```php
/**
 * Kontrola, zda dotaz na soubor je podán skrze HTTP_X_REQUESTED_WITH
 */
public static function isXmlHttpRequest()
{

    if (!isset($_SERVER['HTTP_X_REQUESTED_WITH']) or strtolower($_SERVER['HTTP_X_REQUESTED_WITH']) !== 'xmlhttprequest') {

        header('HTTP/1.1 403 Forbidden');
        exit();
    }
}
```

Obrázek 49 – Funkce *isXmlHttpRequest* v *ActionSecurity*

```php
/**
 * Generace CSRF tokenu
 */
public static function generateCsrfToken()
{

    $csrfToken = bin2hex(random_bytes(32));

    return $csrfToken;
}
```

Obrázek 50 – Funkce *generateCsrfToken* v *ActionSecurity*

6.7 Testovací prostředky a postupy

V této kapitole budou představeny nástroje a budou nastíněny průběhy testování aplikace a databáze. Pro testování bylo využito několik nástrojů, které jsou představeny níže v podkapitolách. Aplikace byla při vývoji používána a testována výhradně v prohlížečích Google Chrome a Brave a nasazená na lokálním Windows zařízení pomocí nástroje Xampp.

6.7.1 Logování

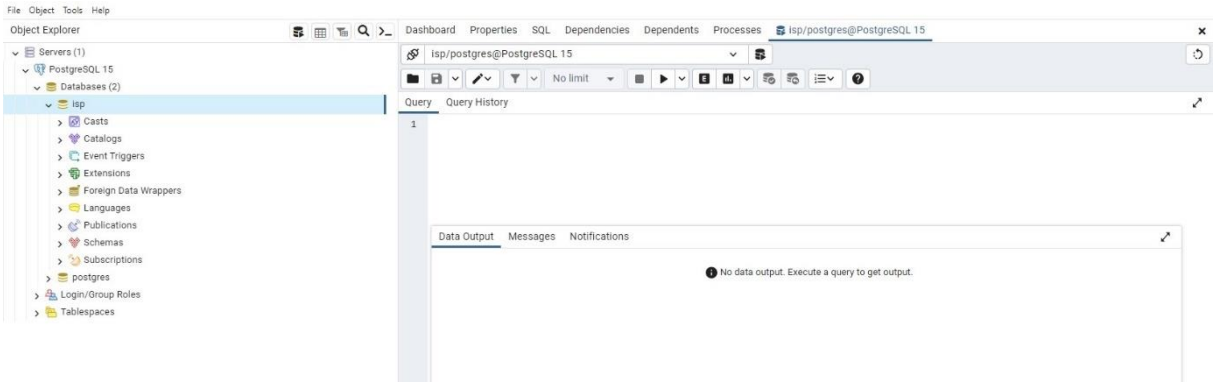
Logování bylo implementováno v rámci PHP kódu pomocí výše představeného balíčku Monolog. Každý odeslaný email, přihlášení a modifikace jakéhokoliv záznamu je zaznamenávána do logu (pokud je to zapnuto v konfiguračním souboru). Díky logování lze výrazně jednodušeji hledat a diagnostikovat chyby ve vygenerovaných dotazech a sledovat historii používání aplikace. Konkrétně se na lokáním zařízení prováděli různé kombinace vstupů, a následně se kontroloval log, kde se hledaly případné chyby.

Všechny soubory určené pro logování vytvářejí informační i chybové hlášky ve chvílích, kdy je logování implementováno v rámci aplikace. Soubory určené pro ukládání logů jsou 3: *db.log, login.log* a *mail.log*. První soubor je určený právě pro zaznamenávání logů informujících o provádění transakcí a případných chybách s příkazy INSERT, UPDATE a DELETE. Vždy je přiložena hláška, která se vrátila z databáze a uživatel, který modifikaci

provedl. Druhý soubor obsahuje informace o přihlašování a odhlašování do aplikace. Poslední soubor *mail.log* zaznamenává informace o zaslaném mailu a případné informace týkající se toho, proč email nebyl odeslán.

6.7.2 PgAdmin 4 a testování databáze

PgAdmin verze 4 je open-source nástroj pro správu, nastavování a testování PostgreSQL databáze. Poskytuje grafické rozhraní pro jednoduchý pohyb v aplikaci, testování dotazů a správu všech dalších záležitostí Postgres databáze (Obrázek 51). (53)



Obrázek 51 - Náhled pgAdmin 4

Při testování aplikace se tento nástroj využíval často pro testování dotazů a rozhodování, jaký způsob dotazu se použije. V tomto nástroji se velmi dobře testují jednotlivé dotazy v kombinaci s SQL příkazem EXPLAIN ANALYZE, který poskytuje možnost, aby databáze vrátila jako výsledek dotazu postup sestaveného plánu pro spuštění daného dotazu. Takovýmto způsobem se například porovnávaly rychlosti jednotlivě sestavených dotazů pro získání dat do přehledu daných objektů na stránce aplikace.

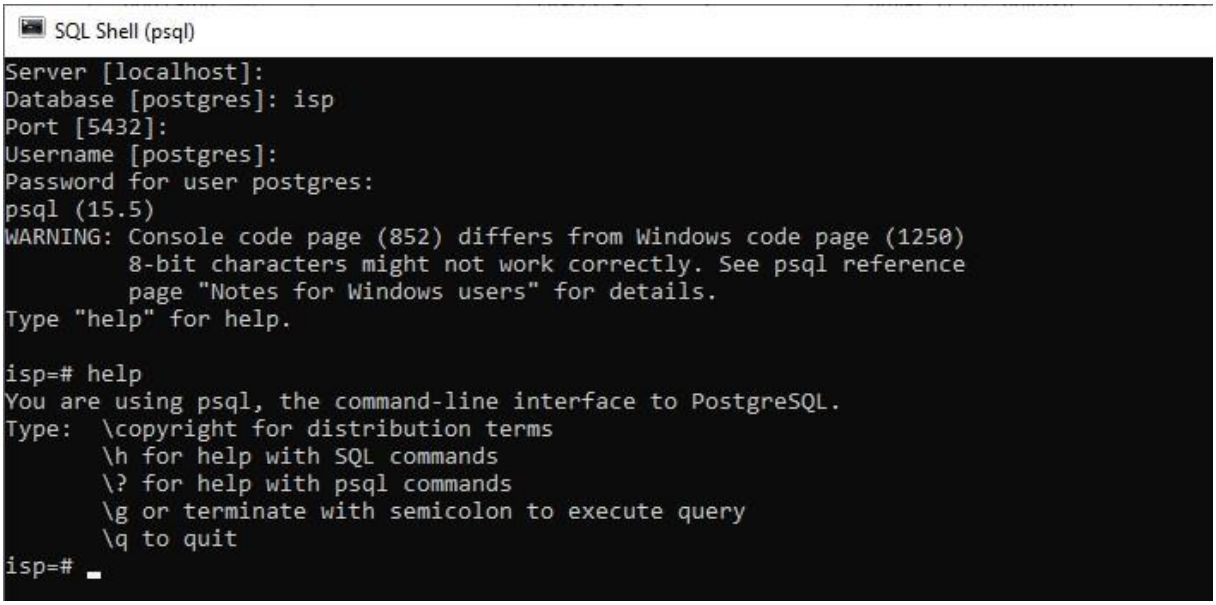
Poté se zjišťovalo, jaké indexy vlastně sama databáze použije, nebo nepoužije na základě jejího vlastního výpočetního rozhodnutí. Ve výsledcích je vidět, že databáze velmi často využívá i jiných technik pro optimalizaci dotazu, a to včetně např. předčasné limitace nebo cachování (Obrázek 52).

	QUERY PLAN text
1	Limit (cost=2.57..76.51 rows=100 width=965) (actual time=0.434..1.934 rows=100 loops=1)
2	-> GroupAggregate (cost=2.57..7397.42 rows=10001 width=965) (actual time=0.433..1.928 rows=100 loops=1)
3	Group Key: service.id, network.id, shaping.id, client.id, status.id, nat.id
4	-> Incremental Sort (cost=2.57..6847.36 rows=10001 width=918) (actual time=0.381..1.226 rows=137 loops=1)
5	Sort Key: service.id, network.id, shaping.id, client.id, status.id, nat.id
6	Presorted Key: service.id
7	Full-sort Groups: 5 Sort Method: quicksort Average Memory: 31kB Peak Memory: 31kB
8	-> Nested Loop Left Join (cost=1.92..6397.35 rows=10001 width=918) (actual time=0.101..1.141 rows=161 loops=1)
9	-> Nested Loop Left Join (cost=1.77..6132.50 rows=10001 width=907) (actual time=0.087..0.968 rows=161 loops=1)
10	-> Merge Left Join (cost=1.61..5882.25 rows=10001 width=789) (actual time=0.079..0.891 rows=161 loops=1)
11	Merge Cond: (service.id = employee_service.service_id)
12	-> Nested Loop Left Join (cost=1.32..5411.93 rows=10000 width=773) (actual time=0.065..0.814 rows=120 loops=1)
13	-> Nested Loop Left Join (cost=1.17..5154.08 rows=10000 width=687) (actual time=0.057..0.699 rows=120 loops=1)
14	-> Nested Loop Left Join (cost=1.02..4889.26 rows=10000 width=661) (actual time=0.047..0.535 rows=120 loops=1)
15	-> Nested Loop Left Join (cost=0.73..4334.50 rows=10000 width=654) (actual time=0.035..0.282 rows=120 loops=1)
16	-> Nested Loop Left Join (cost=0.45..657.59 rows=10000 width=643) (actual time=0.027..0.111 rows=120 loops=1)
17	-> Index Scan using service_pkey on service (cost=0.29..407.29 rows=10000 width=565) (actual time=0.013..0.049 rows=120 loo...
18	-> Memoize (cost=0.16..0.18 rows=1 width=82) (actual time=0.000..0.000 rows=1 loops=120)
19	Cache Key: service.status_id

Obrázek 52 - Část výsledku EXPLAIN ANALYZE

Dále se pomocí pgAdmin rozhodovalo, zda se použije fulltextové řešení vyhledávání poskytnuté v rámci PostgreSQL nebo, zda se využije kombinace podmínek pro jednoduché sestavení dotazu. Rozhodlo se pro druhé řešení, kvůli jeho jednoduššímu nasazení, menší náročnosti na databázi (např. menší potřeba indexování) a 100% přesnosti vyhledávání zadaného řetězce ku jednotlivým polím. Výhodou využití fulltextového řešení by byla možnost vyhledávání na základě podobností a větší možnosti nastavení.

Alternativou pro nástroj pgAdmin je využití obyčejné příkazové řádky v prostředí Windows či Linux. Ve Windows konkrétněji SQL Shell (Obrázek 53). Ten nabízí jednoduché možnosti správy, testování a nastavování, které může být sice méně uživatelsky přívětivé, ale za to mnohem rychlejší. Pomocí tohoto nástroje se například testovalo, zda se nepřidávají redundantní záznamy, ještě, než bylo implementováno logování.



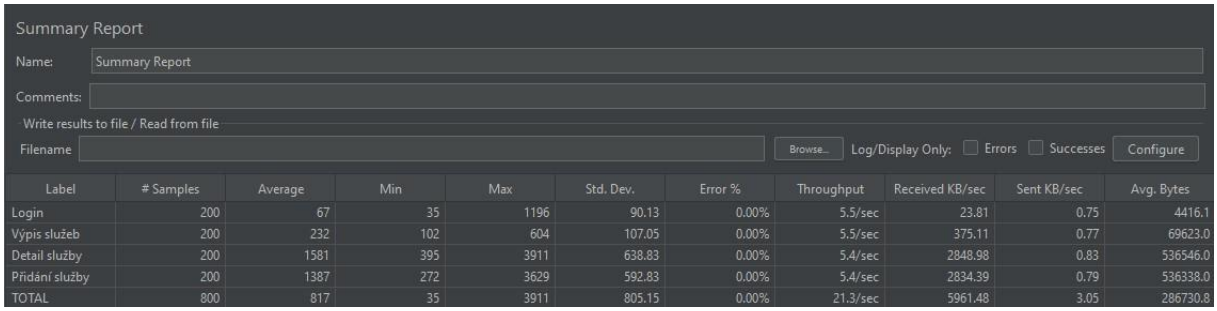
Obrázek 53 - Náhled SQL Shell ve Windows

6.7.3 Apache JMeter

Apache Jmeter je nástroj pro zátěžové testován. V aplikaci lze nastavit různá velikost zátěže a testovat, tak limity systému. Nástroj funguje tak, že simuluje skupiny uživatelů, které posílají dotazy na server, a poté vrací uživateli aplikace statistiky. Nástroj je k dispozici ke stažení na stránce https://jmeter.apache.org/. (54)

Pomocí tohoto nástroje byla aplikace testována zátěžovým testem při kterém 20 nasimulovaných uživatelů zasílalo, každé 4 sekundy dotaz na 4 různé stránky aplikace, a to 20krát za sebou (Obrázek 54). Výsledkem byla v průměru lehce zvýšená doba odezvy stránky (Obrázek 55).

Obrázek 54 – Nastavení testu Apache JMeter



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login	200	67	35	1196	90.13	0.00%	5.5/sec	23.81	0.75	4416.1
Výpis služeb	200	232	102	604	107.05	0.00%	5.5/sec	375.11	0.77	69623.0
Detail služby	200	1581	395	3911	638.83	0.00%	5.4/sec	2848.98	0.83	536546.0
Přidání služby	200	1387	272	3629	592.83	0.00%	5.4/sec	2834.39	0.79	536338.0
TOTAL	800	817	35	3911	805.15	0.00%	21.3/sec	5961.48	3.05	286730.8

Obrázek 55 – Výsledky testu Apache JMeter

7 Uživatelská dokumentace

Webová aplikace *ISP* Commander má sloužit jako jednoduchý a přehledný systém pro zaznamenávání klientů a jejich internetových služeb ve světě ISP. Je vytvořena na míru, jako interní systém pro ISP Presotnet a bude se nadále vyvíjet a posouvat. V budoucnu by do něj měli přibýt funkce jako je například automatické nastavování síťových zařízení od výrobce Cisco.

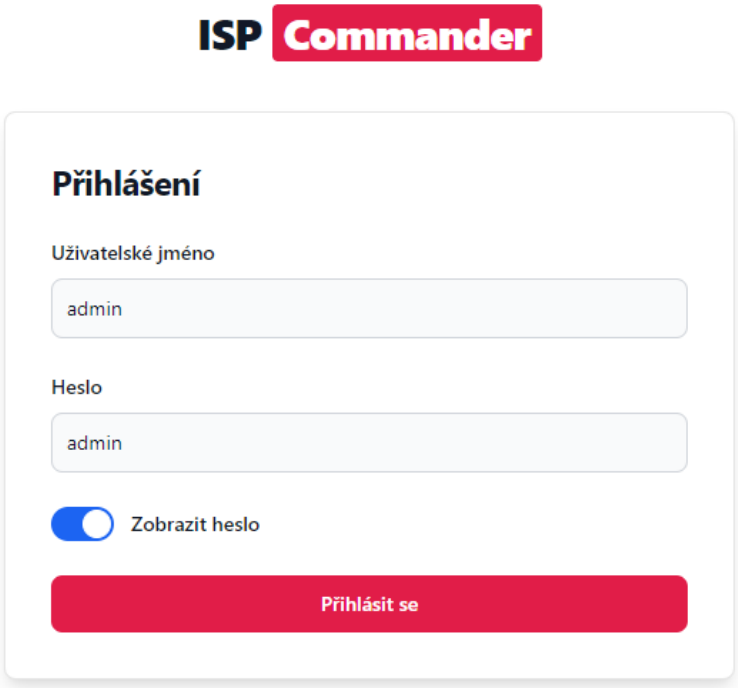
Aplikace je vytvořena výhradně pro techniky, kteří pracují u poskytovatele internetu. Proto se u uživatele předpokládá znalost základních síťovích pojmů a technologií, které jsou spojené s prací u ISP. Těmito základními pojmy jsou myšleny např. síť, IP adresa nebo NAT.

Níže v kapitolách je popsáno, jak pracovat s aplikací, jak se do ní přihlásit, jak vytvořit uživatelský účet a tak dále. Pro testovací účely aplikace běží veřejně nasazená na serveru. Přihlašovací údaje a další důležité údaje k demoverzi aplikace jsou k nalezení v příloze (viz. Příloha A).

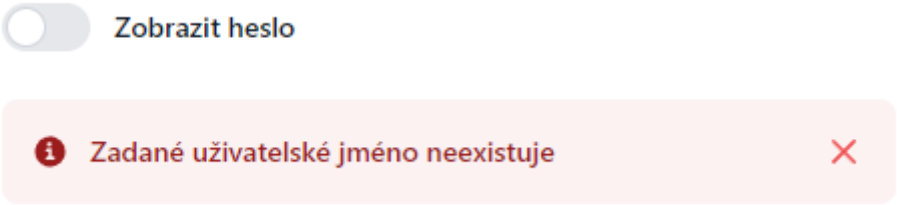
7.1 Účty a přihlášení

Prvotní přihlášení do aplikace probíhá jednoduše zadáním základních přihlašovacích údajů *admin*/*admin* (v demoverzi je heslo změněné). Vytvoření tohoto účtu je totiž součástí inicializačního skriptu pro vytvoření databáze.

Přihlašovací stránka je velmi jednoduchá. Na prostředku stránky se nachází okénko s formulářem, kam se vyplňují přihlašovací údaje (Obrázek 57). Dále se pod vstupem pro heslo nachází tzv. toggle, který umožňuje zadané heslo zobrazit. Pokud se přihlašovací údaje zadají špatně, tak se hned pod tímto togglem objeví informace s chybou (Obrázek 56).

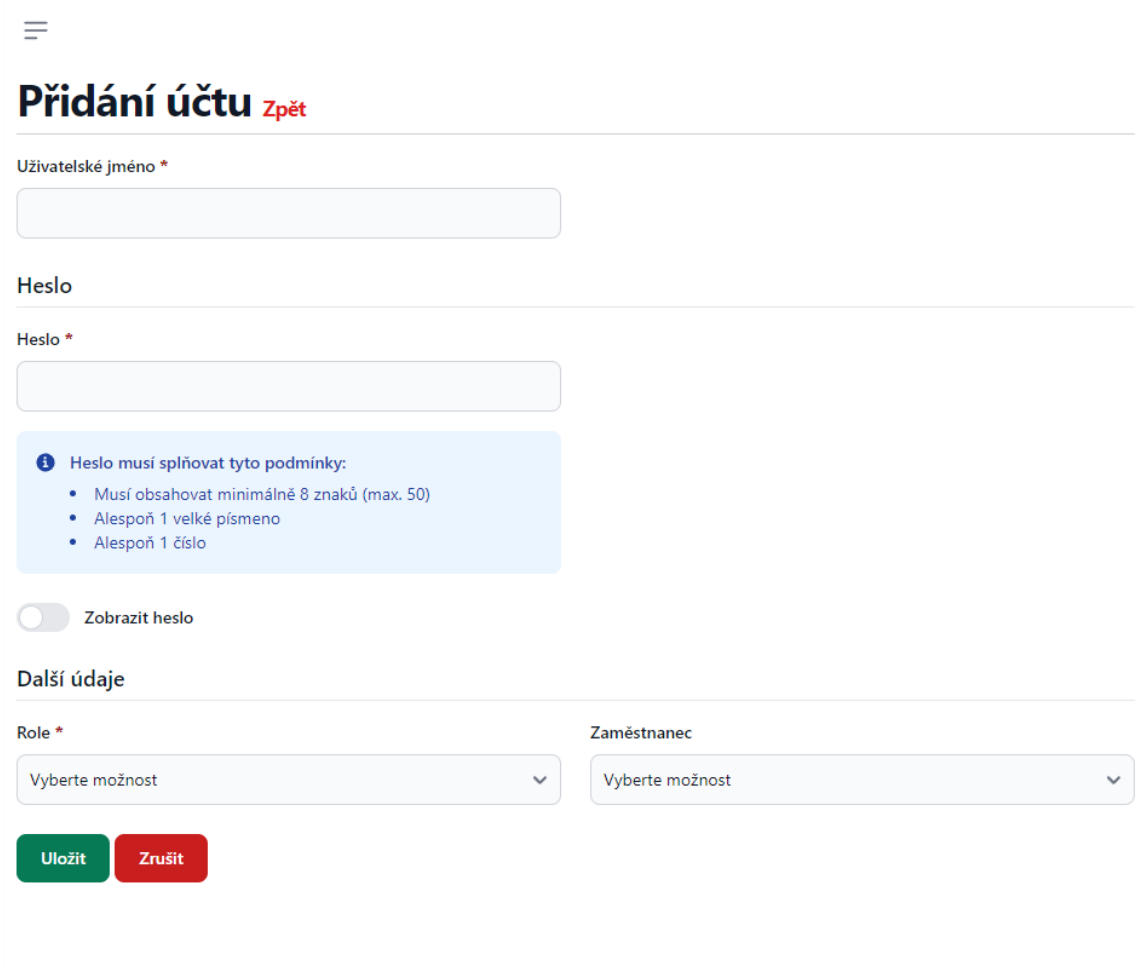


Obrázek 57 – Přihlašovací okno



Obrázek 56 – Chyba při přihlašování

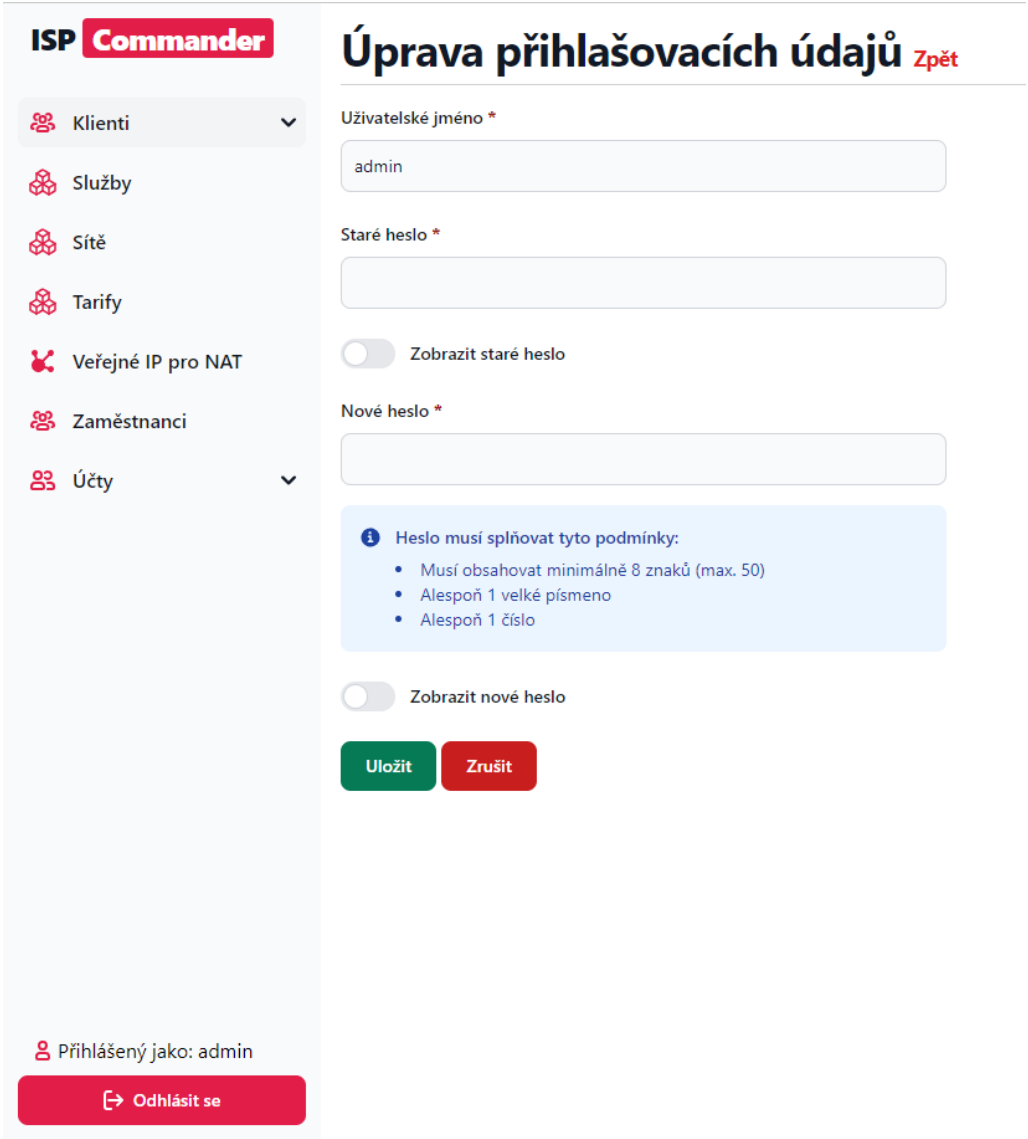
Po přihlášení se objevíte v přehledu klientů. Po levé straně stránky se poté nachází celé menu. Pro přidání nového účtu stačí kliknout v menu na *Účty -> Přehled účtů*, a poté nahoře na zelené tlačítko s plusem. To vás přenese na formulář pro přidání účtu (Obrázek 58), kde stačí vyplnit nové přihlašovací jméno, heslo (dle předepsaných pravidel) a vybrat roli. Dále můžete k danému účtu přiřadit zaměstnance. To ale není povinné.



Obrázek 58 – Přidání účtu

Odhlášení je jednoduché, a to pomocí tlačítka *Odhlásit* se, které se nachází vlevo dole v menu (Obrázek 59). Po vytvoření uživatele a následném odhlášení, je možné se ihned přihlásit jako nový uživatel. Aplikace vás také může sama odhlásit při dlouhé neaktivitě.

Heslo je možné změnit pouze pro účet, na kterém je uživatel právě přihlášený. Pro detail svého účtu stačí kliknout dole v menu na nápis *Přihlášený jako:* -- nebo rozkliknout v menu záložku *Účty* a vybrat dále *Můj Účet*. To vás přenese do detailu účtu, kde by se nahoře mělo nacházet tlačítko s nápisem *Upravit heslo*. Po kliknutí se objevíte ve formuláři pro úpravu přihlašovacích údajů, kde lze heslo upravit pomocí vyplnění starého, a poté nového hesla. Heslo opět musí splňovat stanovené požadavky. (Obrázek 59).



Obrázek 59 – Menu a úprava hesla

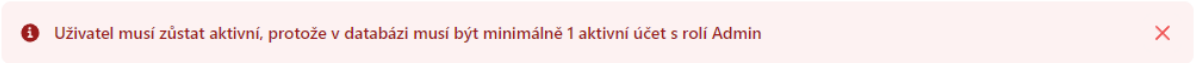
Jak se obecně jednotlivé záznamy přidávají, upravují, mazají, filtrují a vyhledávájí bude představeno až dále v této dokumentaci. Hned po kapitole, kde budou představeny role.

7.1.1 Role

Každý účet má svou přiřazenou roli. V této aplikaci se nachází v základu přesně čtyři role. První role s nejmenšími oprávněními je *Viewer*. Pokud účtu přiřadíte tuto roli tak jediné, co bude moci přihlášený uživatel dělat je číst záznamy a pohybovat se napříč nimi. Další dvě role jsou *Editor* a *Editor+*. Tyto role mohou obě přidávat a upravovat záznamy. Hlavní rozdíl je to, že *Editor+* může záznamy i mazat.

Poslední rolí s největšími oprávněními je potom *Admin*. Ten jako jediná role může spravovat uživatele. To znamená, že je může přidávat, odebírat (deaktivovat) a upravovat, dle libosti. Jediné omezení je v tom, že se v databázi vždy musí nacházet minimálně 1 účet s touto rolí, který je aktivní. A to je ohlídané pomocí kontroly, když se aktuálně přihlášený uživatel pokusí deaktivovat posledního aktivního admina, vrátí se mu upozornění (Obrázek 60). To samé platí, pokud se u daného účtu pokusíte změnit roli.

Hlavní důvod, proč byla zvolena deaktivace účtů, místo jejich úplného smazání je to, že když přihlášený uživatel pod daným účtem vytvoří, či modifikuje záznam, tak jsou tyto data i s jeho uživatelským jménem a aktuálním časovým razítkem napárována přímo k záznamu. Pokud by se tedy účet úplně smazal mohlo by to porušit integritu těchto dat. Tyto pole jsou vysvětlena v rámci kapitoly Konkrétní záznamy a použití.



Obrázek 60 – Upozornění (deaktivace uživatele)

7.2 Práce se záznamy

V rámci aplikace je možné se záznamy manipulovat formou přidávání, upravování a odstraňování. Dále lze v přehledech daných objektů stránkovat, vyhledávat a u některých i filtrovat. To umožňuje jednoduchou a zároveň pokročilejší práci se záznamy.

Jako záznam se nebere vždy jen jeden záznam v tabulce databáze. V rámci aplikace tím může být myšlena kombinace záznamů z různých tabulek v databázi, které svým spojením (kombinací) vytvoří jeden záznam v aplikaci. Například záznam ve službě v aplikaci je tvořen ze záznamů z několika tabulek v databázi jako jsou databázové tabulky *service*, *address*, *network* a tak dále. V aplikaci jsou poté zkombinovány do jednoho „záznamu“ pro přívětivější použití a manipulaci.

7.2.1 Přidání záznamu

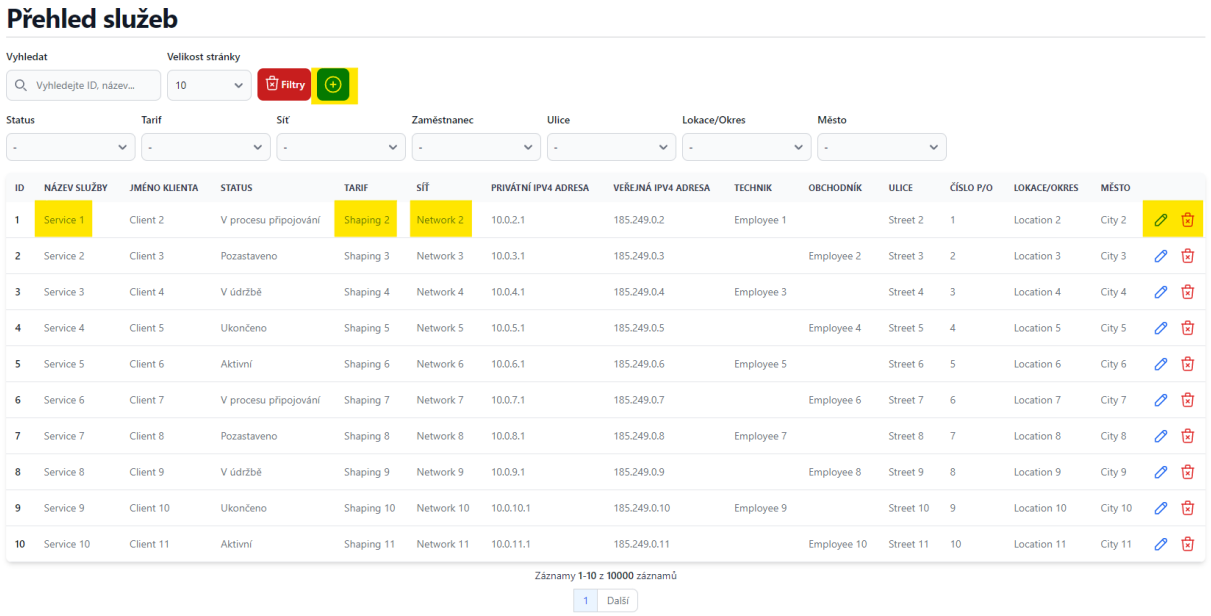
Pro přidání záznamu do jakékoliv tabulky stačí postupovat stejně jako při přidávání nového účtu. Stačí se skrze menu dostat do přehledu objektu, který chcete přidat (jako například služba, klient nebo účet), a poté kliknout na zelené tlačítko s ikonou plus, které se nachází nahoře na stránce (Obrázek 61). To vás přesune na formulář pro přidání daného objektu, kde už vyplníte potřebné informace v rámci daného formuláře. Vstupy, které se musí povinně vyplnit, jsou označeny červeným znakem *.

7.2.2 Modifikace a detail záznamu

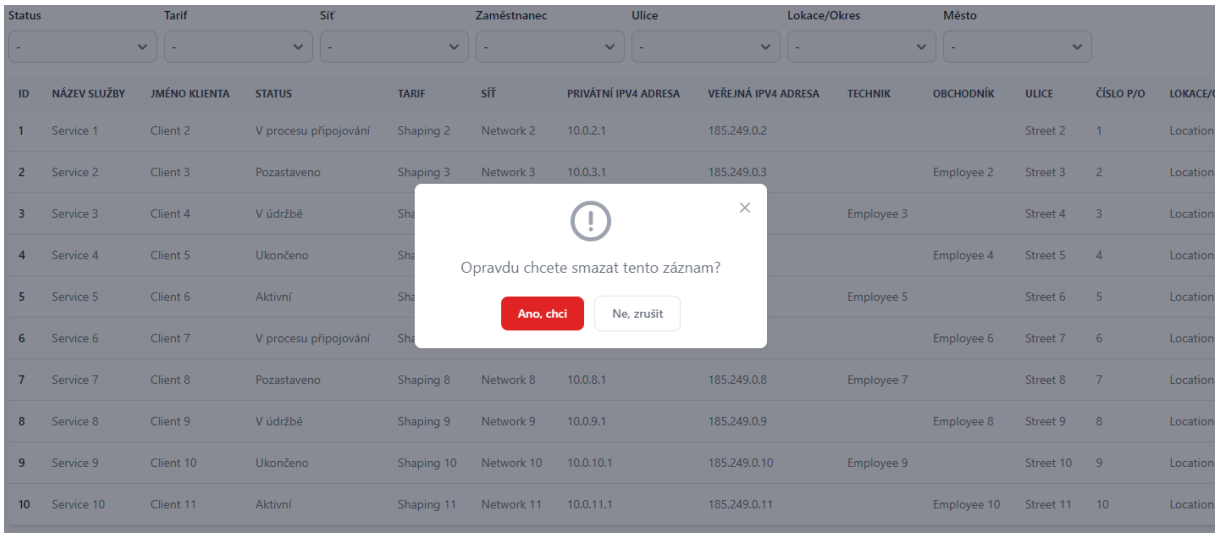
Zobrazení detailu záznamu je v rámci aplikace stejné jako zobrazení formuláře pro upravení záznamu. Obě věci se totiž zobrazují v rámci stejného předvyplněného formuláře, a proto se tento formulář obecně nazývá *Detail*. Do zobrazení detailu záznamu se vždy dostává skrze přehled daného objektu, kde jsou jednotlivé záznamy zobrazeny v tabulce. Pro jeden detail záznamu, pak už stačí kliknout buďto na název/jméno daného záznamu nebo na tužku, která se nachází úplně vpravo v tabulce v každém záznamu. Z přehledu se můžete dostat i na detail jiných objektů, které se zobrazují v tabulce. U přehledu služeb je to například pole *tarif* a *síť* (Obrázek 61).

V detailu záznamu se nahoře vždy nachází název objektu, kterého se týká daný detail, hned vedle se poté nachází červené tlačítko *Zpět*. Po kliknutí na toto tlačítko se objevíte zpět tam, odkud jste se dostali na daný záznam. To samé platí pro červené tlačítko zrušit, které se nachází úplně na spodku formuláře. Zde se také nachází zelené tlačítko *Uložit,* které potvrdí (odešle) daný formulář na server.

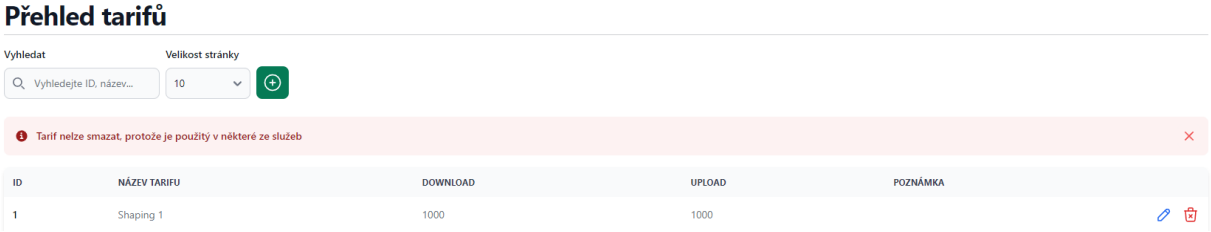
Mazání záznamu je také velmi jednoduché, stačí kliknout na červenou ikonu koše, která se nachází úplně vpravo vedle ikony tužky pro zobrazení detailu. Po kliknutí na tuto ikonu se ještě zobrazí tzv. modál (Obrázek 62), který se zeptá, zda opravdu daný záznam chcete odstranit. Některé záznamy přesto odstranit nelze, a to z důvodu, že se procesně používají v jiném záznamu. To platí například pro sítě a tarify, které jsou použité v některé ze služeb (Obrázek 63). Nebo je lze pouze deaktivovat.



Obrázek 61 – Přehled služeb (modifikace záznamů)



Obrázek 62 - Modál (smazání záznamu)

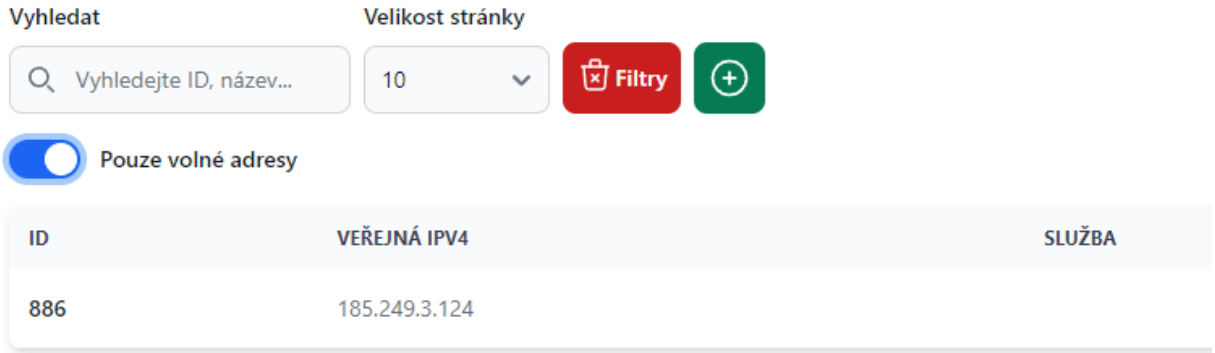


Obrázek 63 – Chyba (mazání záznamu)

7.2.3 Vyhledávání, filtrování a stránkování

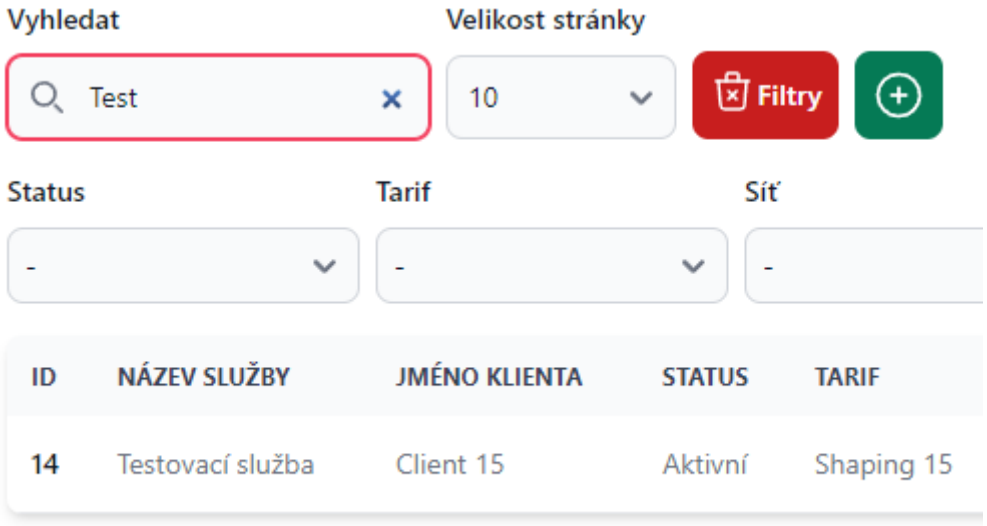
Vyhledávání je velmi jednoduché. Stačí do vyhledávače nahoře na stránce napsat, co chcete najít a stránka automaticky aplikuje vyhledávání (Obrázek 66). Vyhledávat lze dle skoro všech věcí, které se nacházejí v tabulce. Vyhledávač nevyhledává na základě podobností, synonym a podobně, ale pouze přesně to, co je zadané ve vyhledávači. Obsah záznamu se vstupu ve vyhledávači nemusí rovnat, ale obsah záznamu musí vstup někde ve svých datech obsahovat. To znamená, že pokud chci například v přehledu služeb vyhledat záznam s názvem *Testovací služba*, stačí do vyhledávače zadat například *Test* a vyhledají se všechny služby, které ve svém záznamu obsahují toto slovo (Obrázek 65). Protože je vyhledávání tzv. živé (hned po vyplnění vyhledávače se začíná vyhledávat), je u této akce přidané automatické zpoždění, které činí cca. 750 ms. Je to z důvodu, aby se začalo vyhledávat až ve chvíli, kdy uživatel dopíše a aplikace se zbytečně vícekrát nedotazovala na výsledky.

Filtrování je možné pouze u přehledu klientů a služeb, kdy se předpokládá, že tyto tabulky budou obsahovat největší počet záznamů a zároveň jsou jejich záznamy složeny z nejvíce informací. Filtry se nacházejí nahoře na stránce hned pod vyhledávání ve formě výběru, a to například konkrétní ulice ve službě či klientovi (Obrázek 66). Filtr se aplikuje hned po jeho výběru a lze ho kombinovat z ostatními filtry a vyhledáváním. Filtry lze také jednoduše odstranit pomocí červeného tlačítka nacházejícího se mezi výběrem velikosti stránky a tlačítkem pro přidání záznamu. Další (také v podstatě filtr) je v přehledu veřejných IP adres, a je reprezentován pouhým togglem a umožňuje, aby se zobrazily pouze volné veřejné IP adresy (Obrázek 64).

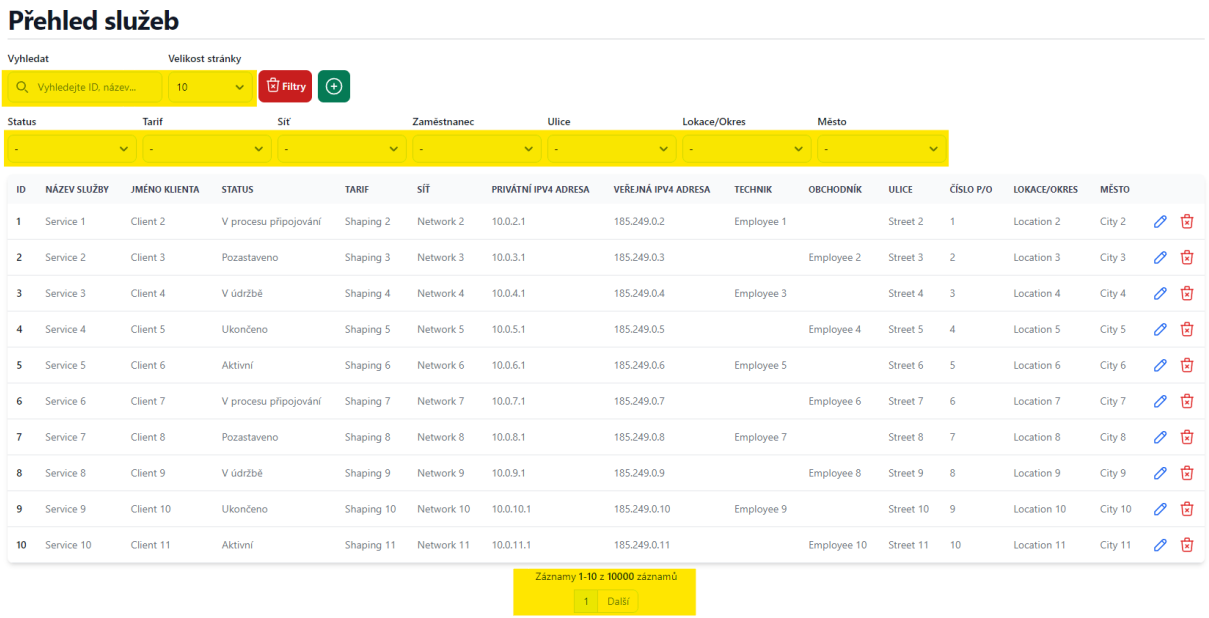


Obrázek 64 – Toggle filtr

Stránkování se skládá z tlačítek, které se nacházejí dole na stránce a výběru, který se nachází nahoře hned vedle vyhledávače. Výběr slouží pro určení velikosti jednotlivých stránek (počtu záznamů na stránku), kde konkrétní možnosti jsou: 10, 25, 50 a 100. Dole se potom nachází stránkování s indikátorem aktuální stránky, aktuální číslo zobrazovaných záznamů, celkový počet záznamů, které odpovídají aktuálně aplikovanému filtru a popřípadě také tlačítka *Další* a *Předchozí* (Obrázek 66).



Obrázek 65 – Příklad vyhledávání



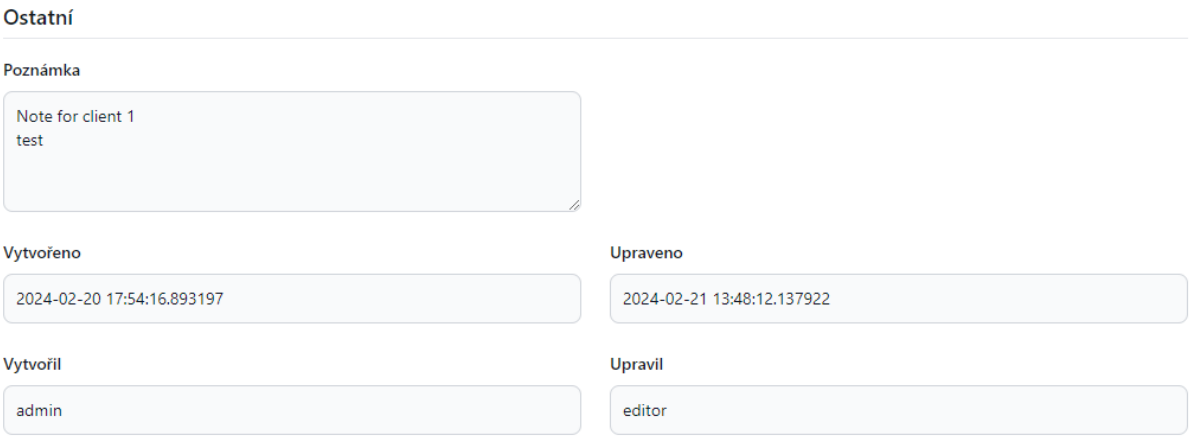
Obrázek 66 – Přehled služeb (vyhledávání, filtrování a stránkování)

7.3 Konkrétní záznamy a použití

V této kapitole bude představena práce s konkrétními záznamy a tabulkami. Bude zde vysvětleno, jak vyplňovat dané pole, jak číst informace ze záznamů a proč jsou v daných záznamech právě konkrétní data. Další podrobnosti, které vysvětlují a odůvodňují jednotlivá pole, jsou vysvětlena také dále v práci v rámci návrhu databáze, která se nachází v technické dokumentaci.

Pole byli prodiskutovány s firmou Prestonet (https://prestonet.cz/), pro kterou je aplikace tvořena. Důvody pro určité nastavení daných polí jsou tedy výsledkem jejich požadavků, kde momentálně hrála roli zásadně jednoduchost, která je myšlena ze strany objemu dat.

Většina objektů (kromě účtů) obsahuje čtyři analytická pole určené pouze pro čtení (read-only). Těmito poli jsou *Vytvořeno*, *Upraveno*, *Vytvořil* a *Upravil*. Tyto pole se nacházejí vždy ve spodní části formuláře v kategorii *Ostatní* (Obrázek 67). Pole *Vytvořeno* a *Upraveno* slouží k zápisu časových razítek (timestamp), které definují, kdy přesně byl záznam vytvořen a kdy byl záznam naposledy upraven. Tato hodnota se vždy automaticky aktualizuje při daném vytváření či modifikaci. Druhá dvě pole jsou *Vytvořil* a *Upravil*. Tyto pole obsahují uživatelské jméno účtu, který záznam vytvořil, či upravil. Platí pro ně to samé jako pro předchozí dvě pole, a to je, že se obě pole automaticky aktualizují.



Obrázek 67 – Analytická pole v záznamu

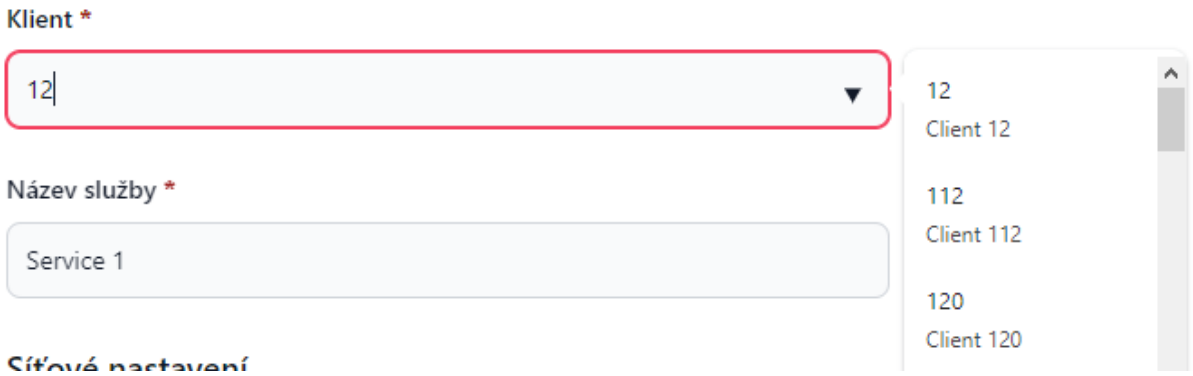
7.3.1 Služba

Služba je nejdůležitějším záznamem ve struktuře aplikace. Jeden záznam služby označuje konkrétní službu poskytovanou konkrétnímu klientovi. To je zapříčiněné tím, že je aplikace koncipovaná pro budoucí vývoj, kdy pomocí jejich záznamů budou automaticky řízena síťová zařízení. Záznam služby obsahuje celkově 28 polí. Právě čtyři z nich jsou read-only analytická pole zmíněna výše.

Povinná pole

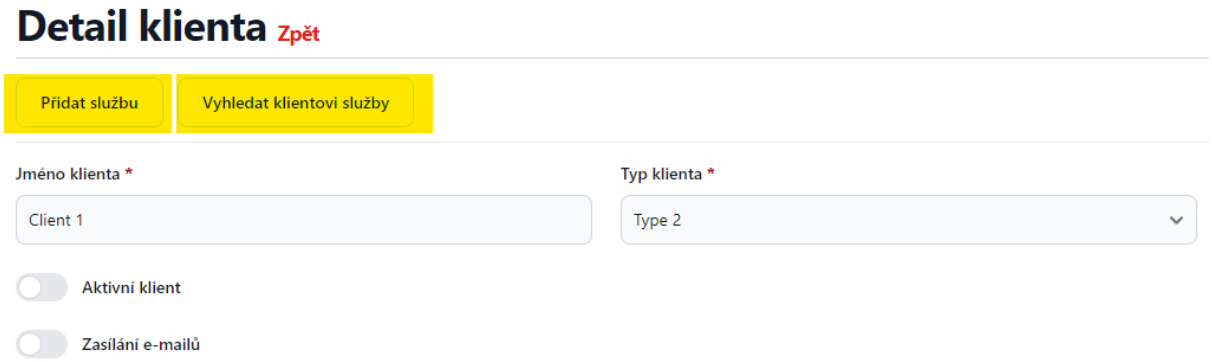
Z toho je 7 nejdůležitějších polí povinných. Povinná pole jsou pouze ta, která jsou procesně nutná pro běh služby. Zbytek už jsou poté pouze reference a dodatečné informace, které v rámci záznamu slouží čistě informativně.

Prvním polem je *Klient*, které je reprezentované ve formě vstupu s datovým listem z možností, které jsou vytvořené pomocí existujících klientů. Datový list (seznam) je stejný jako obyčejný textový vstup, ale prohlížeč nabízí možnosti, ve kterých lze vyhledávat. Validní vstup pro toto pole je ID klienta, proto se musí vždy vybrat možnost, kterou aplikace v rámci datového listu nabídne nebo přímo vyplnit ID potřebného klienta. Výběr (select) se nepoužívá, protože je dost pravděpodobné, že v tabulce klientů bude velký objem záznamů (Obrázek 68).

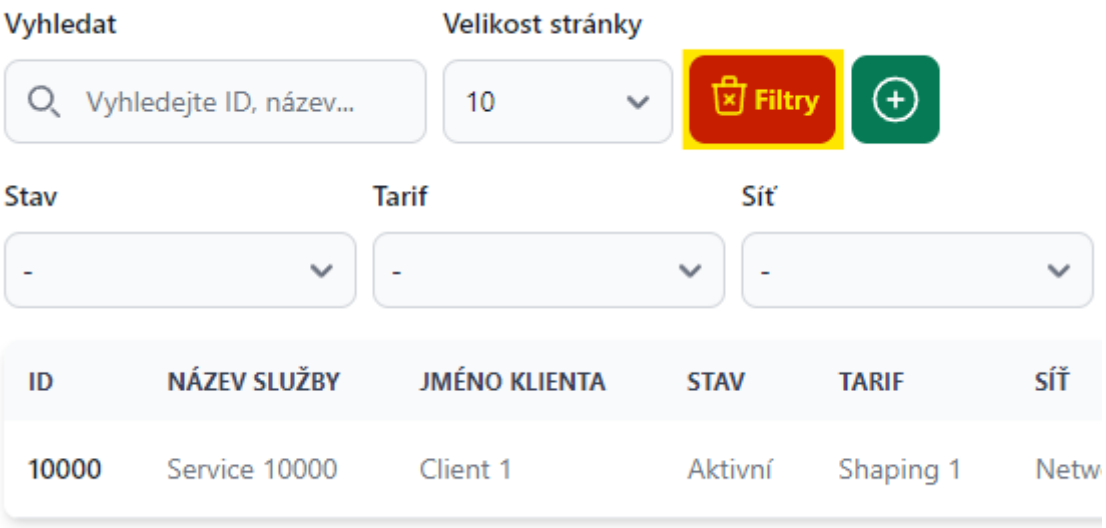


Obrázek 68 – Výběr klienta u služby

Pokud je klientů velký počet je lepší a přehlednější využít tlačítka pro přidání služby nacházejícího se nahoře v detailu klienta (Obrázek 69). Vedle tlačítka pro přidání služby se také nachází tlačítko *Vyhledat klientovi služby*. To vás po kliknutí přesměruje na přehled služeb, kde bude aplikovaný filtr pouze pro daného klienta. Tento filtr se na stránce přímo nenachází ve formě výběru (selectu), proto pokud chcete filtr pouze pro daného klienta odebrat, použijte červené tlačítko pro odstranění filtrů (Obrázek 70).



Obrázek 69 – Přidání a vyhledání služby z detailu klienta



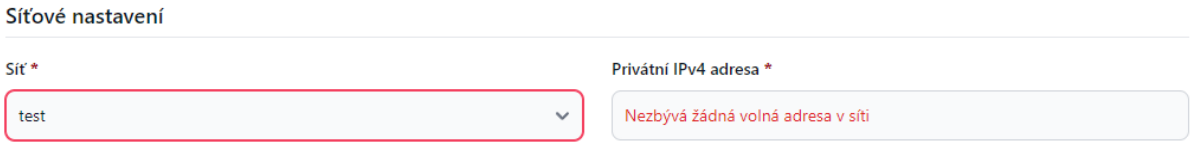
Obrázek 70 – Tlačítko 'Vyhledat klientovi služby'

Druhým polem je samotný *Název služby*. Toto pole je jednoduchý řetězec o maximálně 50 znacích. Toto pole jednoznačně neidentifikuje službu, protože není unikátní pro každý záznam. Je tomu tak, protože v případě, že jsou hodnoty unikátní, tak uživatel aplikace začne často používat neatomická data a začne do tohoto údaje vyplňovat nesmysly, jen aby mu aplikace dovolila záznam přidat. *Název služby* se může vyplňovat například kombinací typu připojení a čísla smlouvy apod. Tímto se aplikace inspirovala dle toho, jak to funguje v rámci již změněné aplikace ISPadmin. Vyplnění tohoto pole musí být důkladné a přehledné, protože se *Název služby* může vyplňovat v rámci automatické emailové komunikace s klientem.

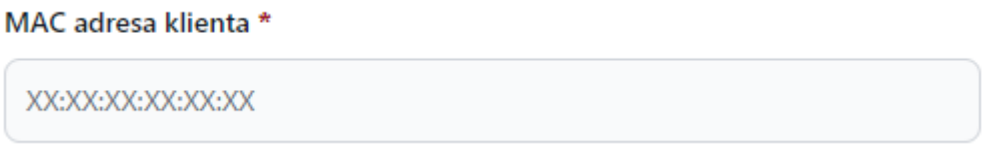
Další pole jsou *Tarif* a *Síť*, které jsou zastoupené pomocí výběru, který se skládá z názvů záznamů těchto objektů. Informace, které se pro danou službu výběrem párují, jsou

specifikována v tom konkrétním záznamu. Popis toho, co tyto záznamy obsahují, se nachází dále v dokumentaci.

Poté jsou zde pole, které definují adresaci pro klientskou bránu (nejčastěji router). To je zařízení, skrze které se klient připojuje k ISP. Těmito poli jsou *Privátní IPv4 adresa* a *MAC adresa klienta*. IP adresa se automaticky vyplní další volnou adresou z vybrané sítě. Předvyplněnou adresu lze samozřejmě změnit, ale musí být validní a nesmí být použitá už v jiné službě nebo jako defaultní brána pro danou síť. Pokud se v síti už nenachází žádná volná adresa, tak je uživatel informován skrze daný vstup (Obrázek 71). MAC adresa se vyplňuje ve formátu, který je ve vstupu vidět jako placeholder (Obrázek 72), jiný formát aplikace nepřijme.

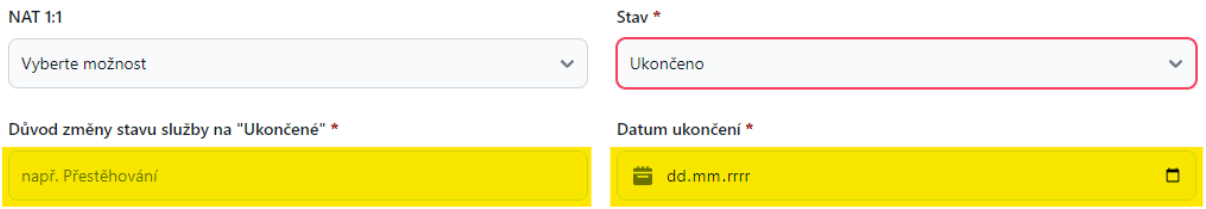


Obrázek 71 – Nezbývá žádná volná adresa



Obrázek 72 – Formát MAC adresy

Posledním povinným polem je *Stav*. Toto pole může nabývat přesně pěti hodnot: V procesu připojování, Aktivní, Pozastaveno, V údržbě a Ukončeno. V případě, že je vybráno některé ze tří posledních možností objeví se dodatečně ještě pole pro udání důvodu změny stavu, které se vyplňuje krátkým odůvodněním o maximálně 30 znacích. U změny stavu na ukončeno se také objevuje pole *Datum ukončení* (Obrázek 73). Pokud se některé z těchto polí zobrazí, tak je vždy povinné ho vyplnit.



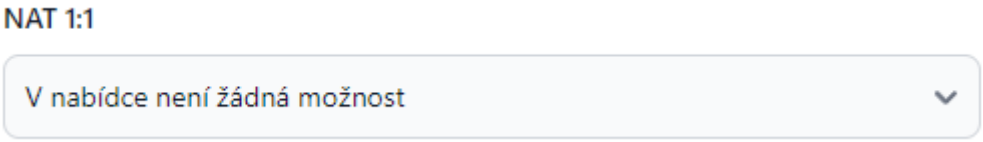
Obrázek 73 – Dodatečné pole k stavu

Dobrovolná pole

Dalším výběrem je také pole *NAT1:1*, toto je ale nepovinné. Je specifické tím, že musí být pro každou službu unikátní, což je indikováno už v názvu (1:1). Aby se tedy zajistilo, že je dané pole vždy unikátní ve výběru se zobrazují pouze nepoužité veřejné IP adresy z tabulky *Veřejné IP pro NAT*, nebo ta, která je momentálně vybrána. Pokud není dostupná žádná veřejná adresa, tak ve výběru se nebude nacházet žádná možnost (Obrázek 74).

Poté je tu kategorie *Další informace*. Ta obsahuje pole *Technik, Obchodník, Den instalace* a *Den poslední údržby*. První dvě pole jsou výběry, které obsahují zaměstnance. Může se do nich vyplnit hlavní osoba (manažer) pro danou činnost (např. vrchní technik). Pole slouží jako případná reference pro uživatele, aby věděl, na koho se má obrátit, pokud potřebuje nějaké konkrétní informace. Ostatní dvě pole jsou poté obyčejná data instalace a údržby, které mohou také poskytnout uživateli potřebný kontext.

Jako poslední jsou pole adresy a poznámka. Adresa se skládá ze sedmi polí, které dohromady poskytují informaci o tom, kde je konkrétní služba poskytována. Adresa je sice velmi důležitá informace, ale jak už bylo zmíněno výše, je nepovinná, protože není povinná pro běh dané služby. Poznámka je poté řetězec o maximálně 300 znacích.

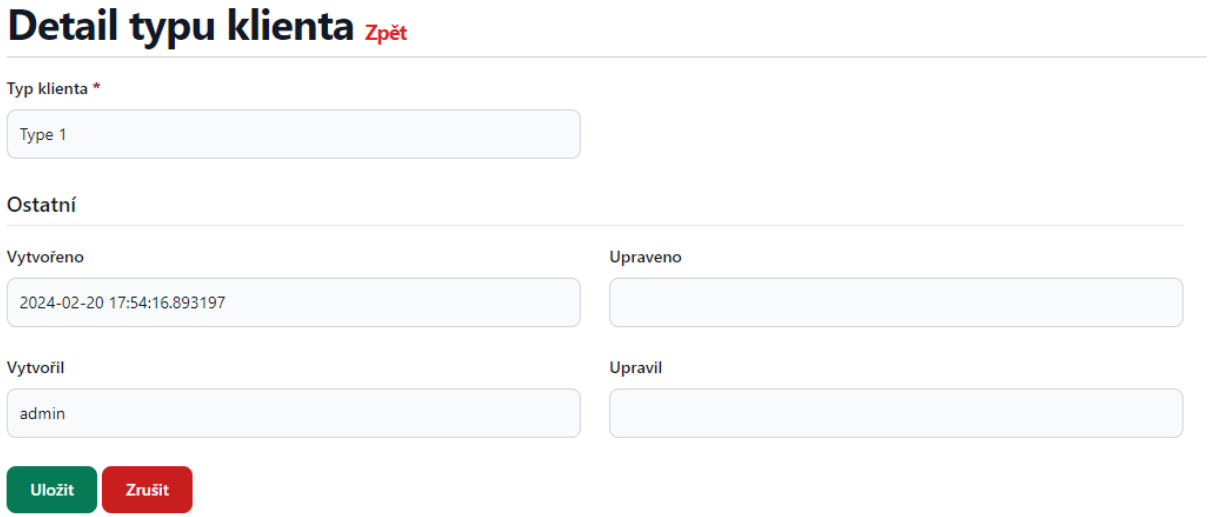


Obrázek 74 – Není dostupná veřejná IP

7.3.2 Klient

Klient je v rámci aplikace konkrétní osoba, firma či instituce, které je poskytována služba. Rozšiřuje dostupné informace ke službě o výhradně kontaktní údaje. Neobsahuje žádné nadbytečné informace, které se týkají např. údajů ke smlouvám a financím. Je tomu tak, kvůli zachování požadované jednoduchosti a specifikovaným požadavkům cílového klienta, pro kterého je aplikace tvořena. Klient tedy neobsahuje finanční údaje a podobně, protože má sloužit výhradně jako dodatečná informace pro službu. Záznam klienta nelze v uživatelském rozhraní aplikace smazat. Opět se jedná o jakési vynucení zachování integrity dat v rámci aplikace. Celkově se ve formuláři nachází 22 (včetně read-only 4 analytických polí).

Povinné pole jsou v záznamu klienta pouze 2. Jsou jimi *Jméno klienta* a *Typ klienta*. Jménem nemusí být myšleno pouze jméno osoby, ale také například název firmy. Jedná se tedy o libovolné pojmenování objektu, kterému je poskytována služba. *Jméno klienta* může mít maximálně 50 znaků a je unikátní pro každý záznam. Další pole je *Typ klienta* a jedná se o jednoduchý výběr. Typy, které se v aplikaci a výběru objeví jsou definovány v přehledu typu klientů, do kterého se dostanete pomocí menu takto: *Klienti -> Typy klienta*. Tato tabulka slouží pro libovolnou kategorizaci klientů, kterou si určí sám uživatel. Typ je reprezentován jako řetězec o maximálně 30 znacích.



Obrázek 75 – Detail typu klienta

Poté ve formuláři následují 2 toggly. Ty určují, zda je daný klient aktivní a jestli se mu smí zasílat automatické emaily. Názvy těchto polí jsou *Aktivní klient* a *Zasílání e-mailů*. Toggle je v podstatě hezčí zaškrtávací okénko, které nabývá hodnoty 0 (nepravda) a 1 (pravda).

Další kategorií jsou kontakty. V klientovi se konkrétně nacházejí kontakty *Primární E-mail*, *Primární Telefon*, *Technický E-mail* a *Technický Telefon*. Tyto kontakty opět slouží jako dodatečné informace, a to výhradně pro technika (uživatele). Proto se v aplikaci také nachází primární kontakty, a poté ty technické. Technické kontakty mají sloužit výhradně jako kontakty na technické oddělení daného klienta v případě, že se jedná o firmu, která takové oddělení má. Telefony jsou povoleny v různých formátech, a ne pouze v tom českém. Aby byla možnost zadávat čísla i z jiných zemích kvůli klientům, kteří nemusí mít české číslo, proto je validace v této formě poměrně omezená.

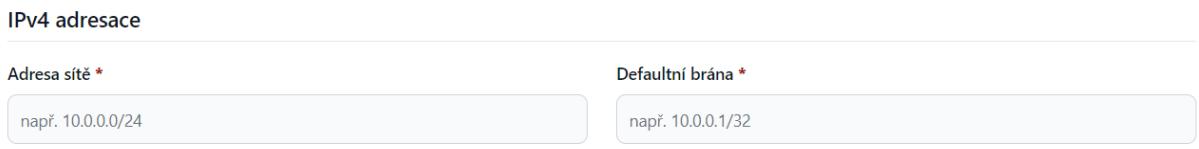
Posledními poli jsou opět pole adresy a poznámka. Tady platí v podstatě to samé jako u adresy ve službě. Zde se akorát vyplňuje trvalé bydliště klienta, či sídlo firmy. Na rozdíl od služby, kde se zadává místo, kde je služba poskytována.

7.3.3 Ostatní objekty

Ostatní objekty v rámci aplikace, které ještě nebyly zmíněny, jsou *Síť*, *Tarif*, *Veřejná IP pro NAT* a *Zaměstnanec*. Data všech těchto objektů se využívají k párování se službami a reprezentují povinné i nepovinné informace o službě. Všechny jsou, jak už bylo zmíněno výše, párované pomocí výběrů, které obsahují jejich unikátní názvy. Ve všech těchto objektech se nachází také pole pro krátkou poznámku, které má maximální velikost 100 znaků.

Síť

Záznam sítě obsahuje technické informace k síti. Obsahuje přesně 5 polí (plus 4 analytická), z nichž jsou 3 povinná (posledním polem je poznámka). Těmi jsou *Název sítě*, *Adresa sítě* a *Defaultní brána*. První pole *Název sítě* je řetězec o maximálně 30 znacích, pomocí kterého lze slovně identifikovat danou síť. *Adresa sítě* je IP adresa, která definuje rozsah adres v rámci dané sítě. Tato adresa je zaznamenávána ve formátu CIDR (Classless Inter-Domain Routing). To znamená, že maska dané sítě je reprezentována prefixem v rámci adresy. Adresa 192.168.1.0 s maskou 255.255.255.0 se tedy do aplikace zadává jako 192.168.1.0/24. Toto pole zároveň nelze upravit z procesních důvodů aplikace. Pokud je tedy potřeba adresu změnit musí se v aplikace vytvořit úplně nová síť. Dále je zde pole *Defaultní brána*, která se musí nacházet v rámci rozsahu sítě (Obrázek 76). A jako poslední pole je validní číslo VLAN, které také musí být unikátní, toto pole je ale dobrovolné.

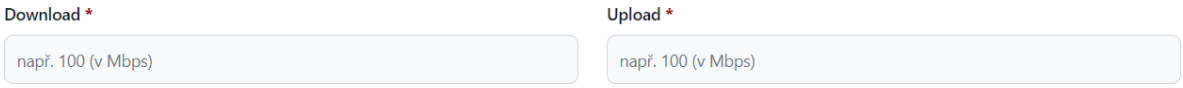


Obrázek 76 – Pole pro IP adresaci v síti

Tarif

Tarif definuje rychlost internetu pro služby. Skládá se ze 3 povinných polí (plus poznámka a analytická pole). Obsahuje pole *Název tarifu*, *Download* a *Upload*. Název funguje stejně jako u sítě a má i stejné parametry, jen identifikuje tarif a ne síť. Ostatní dvě pole definují

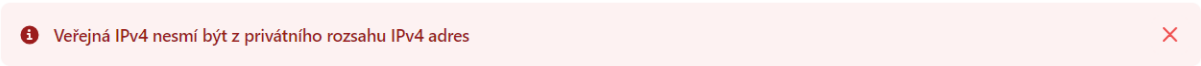
rychlost stahování a nahrávání pro daný tarif. Jsou zadávány pomocí čísla, které reprezentuje rychlost v Mbps/s (Obrázek 77).



Obrázek 77 - Pole pro definici rychlosti tarifu

Veřejná IP pro NAT

Jak už název *Veřejná IP pro NAT* napovídá, tak tento objekt se používá pro párování veřejné IP adresy ke službě. Nachází se ve vlastní tabulce výhradně pro přehlednost, a aby se mohli veřejné IP adresy vložit do systému předem, a poté se až mohly párovat se službami. Formulář obsahuje pouze jedno povinné pole *Veřejná IPv4* (opět také poznámku a analytická pole). Tato adresa musí být unikátní (v rámci systému) a nesmí být z delegovaných rozsahů, jako jsou například privátní IP adresy (Obrázek 78). Dále je zde také název služby, která používá danou adresu. Název služby sice není unikátní, ale po kliknutí na daný název v přehledu se dostanete na její detail.



Obrázek 78 – Chyba při použití delegované adresy

Zaměstnanec

Zaměstnanec jako objekt v rámci aplikace slouží také jako doplnění informací ke službě. Tedy pro přiřazení technika a obchodníka pro konkrétní službu. Dále lze napárovat zaměstnance na účet, to lze udělat skrze záznam účtu. Obsahuje celkově 5 polí (opět také poznámku a analytická pole), z nichž jsou 3 povinná. Těmi jsou *Jméno pro zobrazení*, *Jméno* a *Příjmení*. Všechny tyto pole jsou řetězce. *Jméno pro zobrazení* má maximálně 50 znaků a funguje jako unikátní název pro identifikaci zaměstnance a objevuje se ve výběrech ve službě ve chvílích, kdy se volí nějaký zaměstnanec. Pole pro celé jméno jsou řetězce, které jsou o maximální délce 30 znaků a nejsou unikátní. Dále se v záznamu nachází pracovní kontakty klienta, a to přesněji *Pracovní E-mail* a *Pracovní telefon*. Formát telefonního čísla je stejně jako u čísel v klientovi poměrně variabilní.

7.3.4 Automatická komunikace s klientem

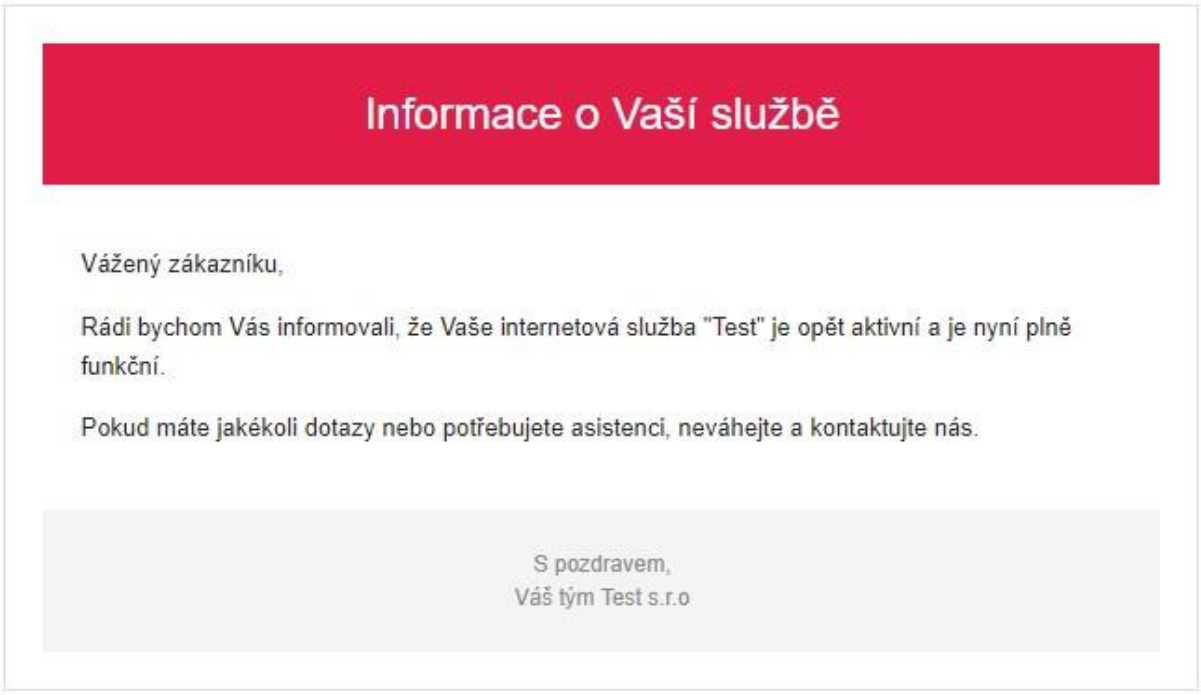
Automatická komunikace s klientem probíhá v rámci aplikace skrze email. Váže se na změnu stavu služby a popřípadě dodatečných polí, které se vážou na stav. To jsou pole pro důvod provedení změny nebo datum ukončení, které se v případě potřeby vyplňují do emailů. Dále se do emailů vyplňuje i pole *Název služby*, pro případ, že jich má daný klient více. V tu chvíli je dobré vyplňovat toto pole ve službě odlišně.

Email je sestavený pomocí HTML a má i alternativní tělo pro případ, že HTML strukturu emailová aplikace klienta nebude podporovat. Jednotlivé emaily pro jednotlivé stavy mají stejnou základní předlohu pro email. Co se u emailů liší, je samotná zpráva. Níže jsou přiloženy příklady přijatých emailů. První se zašle při změně stavu z jakéhokoliv z posledních 3 stavů zpět na *Aktivní* (Obrázek 79). Druhý se zašle v případě, že se stav změní na *V procesu připojení*. Nejčastěji se tento stav používá ve chvíli, kdy se vytváří nová služba (Obrázek 80).

Pro zaslání emailu se musí splnit několik podmínek. Pokud se tyto podmínky nesplní systém žádný email klientovi neodešle. Podmínek je několik a nezáleží pouze na tom, co je nastavené v uživatelském prostředí aplikace. Těmito podmínkami jsou:

- Automatická komunikace musí být povolena v konfiguračním souboru
- Klient musí být aktivní
- Klient musí mít povolené zasílání emailů
- Klient musí mít vyplněné pole *Primární E-mail*

Pokud jsou splněny všechny podmínky pořád se může stát, že se email neodešle. A to v případě, že vyplněný *Primární E-mail* obsahuje neexistující emailovou adresu. V tom případě vám přijde upozornění na emaily, skrze který se odesílají emaily pro automatickou komunikaci.



Obrázek 79 – Email reaktivace služby

Obrázek 80 – Email, služba v procesu připojování

8 Závěr

Cílem tohoto projektu bylo vyvinout funkční webovou aplikaci pro zaznamenávání klientů a jejich internetových služeb, kterou v praxi využije firma Prestonet. V projektu byly nejdříve popsány odborná témata týkající se systémů pro ISP, relačních databází a webových technologií. Poté byla popsána samotná struktura aplikace v rámci technické a uživatelské dokumentace.

Všechny cíle stanované v úvodu byly splněny a do aplikace byly přidány i další dodatečné funkce. Aplikace umožňuje zaznamenávání důležitých technických a kontaktních dat o službách a klientech. Nabízí vyhledávání, filtrování a stránkování napříč daty, a to velmi jednoduchým a přehledným způsobem. Dále automatickou komunikaci se zaznamenávanými klienty, logování, validaci vstupů, automatické předvyplnění IP adresy při výběru sítě pro službu a mnohem více. Velkou výhodou aplikace je její jednoduchost a menší objem informací, který zajišťuje větší přehlednost a konkrétnější užití aplikace.

Pro vytvoření aplikace byly využity technologie HTML, CSS, JavaScript a PHP verze 8.2. Pro databázi poté PostgreSQL verze 15. Aplikace se bude nadále vyvíjet a budou se do ní přidávat nové funkce. V budoucnu bude aplikace například sloužit jako nástroj pro automatickou konfiguraci síťových zařízení.

9 Zdroje

1. **Gillis, Alexander S.** ISP (internet service provider). *Web TechTarget.* [Online] TechTarget. [Citace: 17. Říjen 2023.] https://www.techtarget.com/whatis/definition/ISP-Internet-service-provider.

2. **CE Colo.** Datacenter: CE Colo. *Web CE Colo.* [Online] CE Colo. [Citace: 17. Říjen 2023.] https://www.cecolo.com/.

3. **Kerner, Sean Michael.** 4G (fourth-generation wireless). *Web TechTarget.* [Online] TechTarget. [Citace: 17. Říjen 2023.] https://www.techtarget.com/searchmobilecomputing/definition/4G.

4. **Priyanka.** Maximizing Profitability in the ISP Industry. *Racknap blog.* [Online] Racknap. [Citace: 31. Říjen 2023.] https://www.racknap.com/blog/strategies-to-maximizing-profitability-in-isp-industry/.

5. **Koďousková, Barbora.** INFORMAČNÍ SYSTÉMY V KOSTCE: ERP, CRM, IMPLEMENTACE. *Web Rascasone.* [Online] Rascasone, s.r.o. [Citace: 21. Říjen 2023.] https://www.rascasone.com/cs/blog/informacni-systemy-erp-crm-implemetace#implementace-informacn-iacute-ho-syst-eacute-mu-a-jeho-zivotn-iacute-cyklus.

6. **Inc., Ubiquiti.** UISP overview. *Web UISP.* [Online] Ubiquiti Inc. [Citace: 14. Říjen 2023.] https://uisp.com/eu/uisp-overview.

7. **Jangid, Roshan.** Introduction to UniFi Ubiquiti Network. *Web Login Radius Inc.* [Online] LoginRadius Inc. [Citace: 14. Říjen 2023.] https://www.loginradius.com/blog/engineering/introduction-to-unifi-ubiquiti-network/.

8. **NET service solution, s.r.o.** ISP admin. *Web ISP admin.* [Online] NET service solution, s.r.o. [Citace: 20. Říjen 2023.] https://ispadmin.eu/.

9. **ISPsystem.** DCImanager. *Web ISPsystem.* [Online] ISPsystem. [Citace: 22. Říjen 2023.] https://ispsystem.com/dcimanager.

10. —. Dokumentace DCImanager. *Web ISPsystem DCImanager dokumentace.* [Online] ISPsystem. [Citace: 22. Říjen 2023.] https://docs.ispsystem.com/dcimanager-admin.

11. **IBM.** What is a relational database? *Web IBM.* [Online] IBM. [Citace: 14. Únor 2024.] https://www.ibm.com/topics/relational-databases.

12. **Oracle.** What is a Relational Database (RDBMS)? *Web Oracle.* [Online] Oracle. [Citace: 14. Únor 2024.] https://www.oracle.com/database/what-is-a-relational-database/.

13. **Inovace VOV.** E-R modelování. *Web VOV ČR.* [Online] ČVUT v Praze, Fakulta elektrotechnická. [Citace: 14. Únor 2024.] https://www.vovcr.cz/odz/tech/393/page16.html.

14. **Rouse, Margaret.** Dictionary: Primary key. *Web Techopedia.* [Online] Techopedia. [Citace: 14. Únor 2024.] https://www.techopedia.com/definition/5547/primary-key.

15. —. Dictionary: Candidate Key. *Web Techopedia.* [Online] Techopedia. [Citace: 14. Únor 2024.] https://www.techopedia.com/definition/21/candidate-key.

16. —. Dictionary: Foreign Key. *Web Techopedia.* [Online] Techopedia. [Citace: 14. Únor 2024.] https://www.techopedia.com/definition/7272/foreign-key.

17. **GeeksforGeeks.** Normal Forms in DBMS. *Web GeeksforGeeks.* [Online] GeeksforGeeks. [Citace: 14. Únor 2024.] https://www.geeksforgeeks.org/normal-forms-in-dbms/.

18. **Upadhyay, Mithlesh.** Boyce-Codd Normal Form (BCNF). *Web GeeksforGeeks.* [Online] GeeksforGeeks. [Citace: 14. Únor 2024.] https://www.geeksforgeeks.org/boyce-codd-normal-form-bcnf/.

19. **Amazon Web Services.** What is SQL (Structured Query Language)? *Web Amazon Web Services.* [Online] Amazon Web Services, Inc. [Citace: 14. Únor 2024.] https://aws.amazon.com/what-is/sql/.

20. **GeeksforGeeks.** SQL | DDL, DQL, DML, DCL and TCL Commands. *Web GeeksforGeeks.* [Online] GeeksforGeeks. [Citace: 14. Únor 2024.] https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/.

21. —. SQL TRANSACTIONS. *Web GeeksforGeeks.* [Online] GeeksforGeeks. [Citace: 14. Únor 2024.] https://www.geeksforgeeks.org/sql-transactions/.

22. **Oyama, Faith.** Advanced Indexing Strategies in PostgreSQL. *Web freeCodeCamp.* [Online] freeCodeCamp. [Citace: 14. Únor 2024.] https://www.freecodecamp.org/news/postgresql-indexing-strategies/.

23. **Ali Khan, Sardar Mudassar.** A Comprehensive Guide to SQL Query Optimization Techniques. *Web DEV Community.* [Online] DEV Community. [Citace: 14. Únor 2024.] https://dev.to/sardarmudassaralikhan/a-comprehensive-guide-to-sql-query-optimization-techniques-1jjk.

24. **PostgreSQL Global Development Group.** About PostgreSQL. *Web PostgreSQL.* [Online] PostgreSQL Global Development Group. [Citace: 14. Únor 2024.] https://www.postgresql.org/about/.

25. **Amazon Web Services.** What is PostgreSQL? *Web Amazon Web Services.* [Online] Amazon Web Services, Inc. [Citace: 14. Únor 2024.] https://aws.amazon.com/rds/postgresql/what-is-postgresql/.

26. **Thakur, Rohan.** Indexes, Views, and Triggers in PostgreSQL. *Web Medium.* [Online] Medium. [Citace: 15. Únor 2024.] https://medium.com/learning-sql/indexes-views-and-triggers-in-postgresql-e88f46f404be.

27. **Amazon Web Services.** What's the Difference Between MySQL and PostgreSQL? *Web Amazon Web Services.* [Online] Amazon Web Services, Inc. [Citace: 14. Únor 2024.] https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/.

28. **Smallcombe, Mark.** PostgreSQL vs MySQL: The Critical Differences. *Web Integrate.io.* [Online] Integrate.io. [Citace: 14. Únor 2024.] https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/.

29. **MySQL.** Introduction to InnoDB. *Web MySQL.* [Online] Oracle. [Citace: 14. Únor 2024.] https://dev.mysql.com/doc/refman/8.0/en/innodb-introduction.html.

30. **Kumar, Dharmedra.** Web Technology. *Web GeeksforGeeks.* [Online] GeeksforGeeks. [Citace: 17. Únor 2024.] https://www.geeksforgeeks.org/web-technology/.

31. **MDN web docs.** An overview of HTTP. *Web MDN web docs.* [Online] Mozilla Corporation's. [Citace: 17. Únor 2024.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

32. **Chai, Wesley a Ferguson, Kevin.** HTTP (Hypertext Transfer Protocol). *Web TechTarget.* [Online] TechTarget. [Citace: 17. Únor 2024.] https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol.

33. **The Internet Society.** RFC 2616. *Web IETF datatracker.* [Online] The Internet Society. [Citace: 19. Únor 2024.] https://datatracker.ietf.org/doc/html/rfc2616.

34. **Cloudflare.** What is SSL? | SSL definition. *Web Cloudflare.* [Online] Cloudflare, Inc. [Citace: 17. Únor 2024.] https://www.cloudflare.com/learning/ssl/what-is-ssl/.

35. **MDN web docs.** HTML basics. *Web MDN web docs.* [Online] Mozilla Corporation's. [Citace: 19. Únor 2024.] https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.

36. —. What is CSS? *Web MDN web docs.* [Online] Mozilla Corporation's. [Citace: 19. Únor 2024.] https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS.

37. **Kolade, Chris.** What is PHP? The PHP Programming Language Meaning Explained. *Web freeCodeCamp.* [Online] freeCodeCamp. [Citace: 17. Únor 2024.] https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/.

38. **Demircioğlu, Atakan.** The Magic Behind PHP. *Web Jotform Tech.* [Online] Medium. [Citace: 17. Únor 2024.] https://tech.jotform.com/the-magic-behind-php-cc6ed3a00cf6.

39. **W3Schools.** PHP - What is OOP? *Web W3Schools.* [Online] Refsnes Data. [Citace: 17. Únor 2024.] https://www.w3schools.com/php/php_oop_what_is.asp.

40. **MDN web docs.** What is JavaScript? *Web MDN web docs.* [Online] Mozilla Corporation's. [Citace: 17. Únor 2024.] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

41. **Bray, Tim.** RFC 8259. *Web IETF datatracker.* [Online] IETF. [Citace: 19. Únor 2024.] https://datatracker.ietf.org/doc/html/rfc8259.

42. **W3Schools.** AJAX Introduction. *Web W3Schools.* [Online] Refsnes Data. [Citace: 24. Únor 2024.] https://www.w3schools.com/js/js_ajax_intro.asp.

43. **The PostgreSQL Global Development Group.** Network Address Types. *Web Postgresql.* [Online] The PostgreSQL Global Development Group. [Citace: 25. Únor 2024.] https://www.postgresql.org/docs/current/datatype-net-types.html.

44. —. Constraints. *Web Postgresql.* [Online] The PostgreSQL Global Development Group. [Citace: 25. Únor 2024.] https://www.postgresql.org/docs/current/ddl-constraints.html.

45. **Composer.** Introduction to Composer. *Web Composer.* [Online] Composer. [Citace: 17. Únor 2024.] https://getcomposer.org/doc/00-intro.md.

46. **Heidi, Erika.** What is Composer? *Web DigitalOcean.* [Online] DigitalOcean. [Citace: 17. Únor 2024.] https://www.digitalocean.com/community/tutorials/what-is-composer.

47. **Bointon, Marcus.** PHPMailer. *Web GitHub.* [Online] GitHub Inc. [Citace: 19. Únor 2024.] https://github.com/PHPMailer/PHPMailer.

48. **Tailwind Labs Inc.** Get started with Tailwind CSS. *Web Tailwind CSS.* [Online] Tailwind Labs Inc. [Citace: 27. Únor 2024.] https://tailwindcss.com/docs/installation.

49. **Sapna2001.** Introduction to Tailwind CSS. *Web GeeksforGeeks.* [Online] GeeksforGeeks. [Citace: 27. Únor 2024.] https://www.geeksforgeeks.org/introduction-to-tailwind-css/.

50. **Flowbite.** Flowbite - Tailwind CSS component library. *Web Flowbite.* [Online] Flowbite. [Citace: 27. Únor 2024.] https://flowbite.com/docs/getting-started/introduction/.

51. **The PHP Group.** PDO Introduction. *Web PHP.* [Online] The PHP Group. [Citace: 28. Únor 2024.] https://www.php.net/manual/en/intro.pdo.php.

52. **Pejša, Jan.** Co je Cross-Site Request Forgery a jak se mu bránit. *Web zdroják.* [Online] Devel.cz Lab s.r.o. . [Citace: 29. Únor 2024.] https://zdrojak.cz/clanky/co-je-cross-site-request-forgery-a-jak-se-branit/.

53. **The PostgreSQL Global Development Group.** pgAdmin 4. *Web pgAdmin.* [Online] The PostgreSQL Global Development Group. [Citace: 26. Únor 2024.] https://www.pgadmin.org/docs/pgadmin4/latest/index.html.

54. **S. Gillis, Alexander.** Apache JMeter. *Web TechTarget.* [Online] TechTarget. [Citace: 26. Únor 2024.] https://www.techtarget.com/searchsoftwarequality/definition/Apache-JMeter.

10 Seznam obrázků

11 Příloha A

Demo aplikace je nasazené přesně tak, jak je to popsané v technické dokumentaci v kapitole pro nasazení aplikace. Jen už je v základu změněno heslo a do aplikace jsou pomocí PostgreSQL skriptu vytvořeny testovací záznamy. Co se týče konfigurace, tak v demu je vše povolené, tím je myšleno konkrétně všechno logování a odesílání emailů klientům. Server běží na infrastruktuře, kterou vlastní Prestonet.

URL: https://vojtapoky.cz/maturita_project/login/

Přihlašovací údaje pro vedoucího

Uživatelské jméno: vedouci

Heslo: cyu3bxj-gbc7XMH0wuw

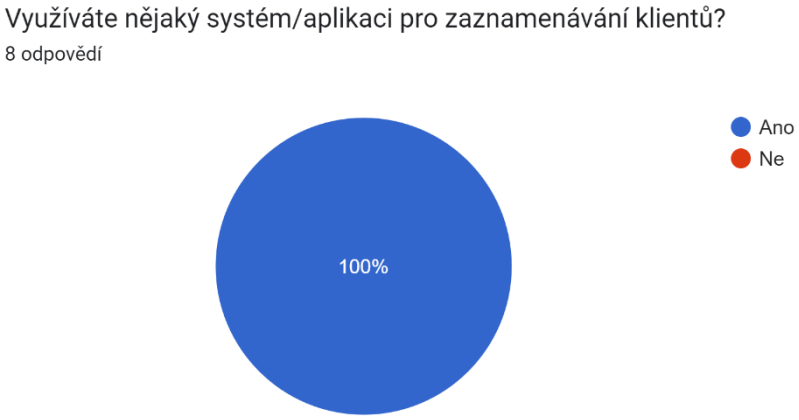
Přihlašovací údaje pro oponenta

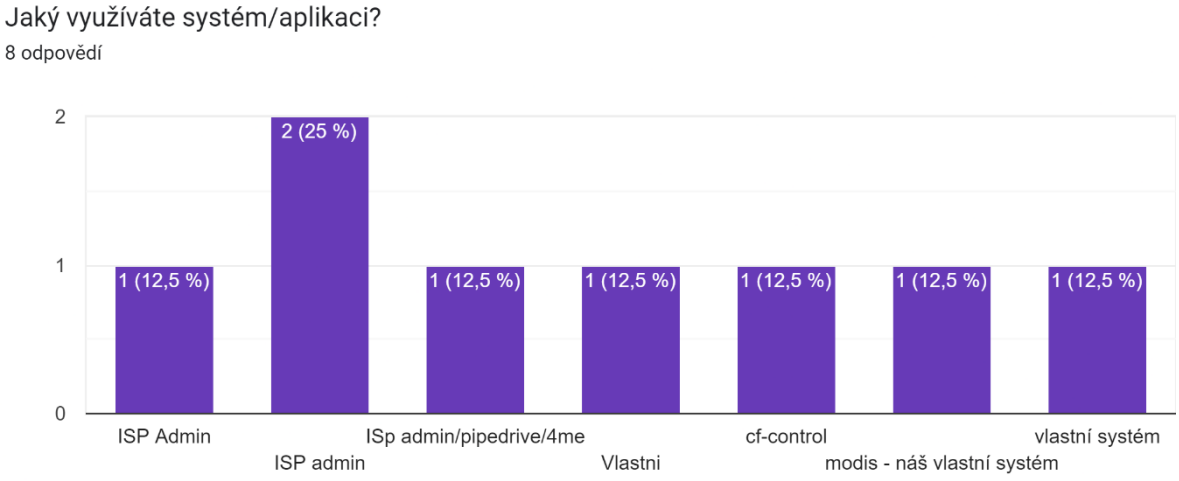
Uživatelské jméno: oponent

Heslo: bnw_hkh!vkz0mqp4VNK

12 Příloha B

Průzkum trhu byl vytvořen v rámci dotazníku, který byl zaslán mezi 40 náhodných ISP sídlících v ČR pomocí jednoduchého emailu. Výsledky odpovědí jsou níže znázorněny ve formě grafů.





Jste s ním spokojený?

8 odpovědí

Kolik byste byli ochotni zaplatit za řešení takovéto webové aplikace? (v korunách)

8 odpovědí

- 50000
- dle dohody a rozsahu práce
- již máme
- Asi né tolik aby se to vůbec vyplatilo dělat
- je již součástí mnou využívané aplikace
- 100.000 Kč - one time fee nebo např. 5000 Kč měsíčně / licence pro 5000 uživatelů
- 10000
- nevím co přesn bude řešit