TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

WEBOVÁ APLIKACE PRO SPRÁVU IPV6 ADRES

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Václav Palík**
Vedoucí práce: doc. RNDr. Pavel Satrapa, Ph.D.

Liberec 2016

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

WEB APPLIACATION FOR IPV6 ADDRESS SPACE MANAGEMENT

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Václav Palík**
Supervisor: doc. RNDr. Pavel Satrapa, Ph.D.

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Václav Palík**
Osobní číslo:	**M14000174**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Webová aplikace pro správu IPv6 adres**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se s problematikou přidělování IPv6 adres a platnými pravidly RIPE NCC.
2. Navrhněte webovou aplikaci pro správu adresního prostoru IPv6.
3. Požadované funkce: grafické znázornění alokací, volitelná úroveň hierarchického pohledu, možnost přidělovat, měnit a rušit alokace, možnost ukládat a načítat data ve formátu XML.
4. Aplikaci implementujte a otestujte.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **cca 50 stran**

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

Seznam odborné literatury:

[1] CASTRO, Elizabeth a HYSLOP, Bruce: HTML5 a CSS3. Brno: Computer Press, 2012. ISBN 9788025137338.
[2] kolektiv autorů: jQuery: Kuchařka programátora. Brno: Computer Press, 2010. ISBN 9788025131527.
[3] SATRAPA, Pavel. IPv6. Praha: CZ.NIC, 2011. ISBN 978-80-904248-4-5.
[4] KOSEK, Jiří. XML pro každého: Podrobný průvodce. Praha: Grada, 2000. ISBN 80-7169-860-1.

Vedoucí diplomové práce: **doc. RNDr. Pavel Satrapa, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání diplomové práce: **20. října 2015**

Termín odevzdání diplomové práce: **16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2015

Prohlášení

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

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

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

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

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

Datum: 16.5.2016

Podpis:

Abstrakt

Tato práce popisuje základní problematiku IPv6 a webových technologií a jejich aplikaci ve správě adresního prostoru IPv6. Hlavním zaměřením této práze je vizualizace bloků adres jako pomůcka pro síťové administrátory ve správě jejich adresního prostoru IPv6.

Klíčová slova: IPv6, webová aplikace, PHP, RIPE NCC, adresní prostor

Abstract

This work describes the basics about IPv6 and web techologies in its application in IPv6 address space management. The main focus of this work is to visualize allocated address blocks in order to help network administrators in managing their IPv6 address space.

Keywords: IPv6, web application, PHP, RIPE NCC, address space

Poděkování

Rád bych poděkoval vedoucímu práce za velice užitečné rady v průběhu práce.

Obsah

1. Úvod....3
2. Co je IPv6?....4
3. PHP 7....12
4. Návrh aplikace....13
5. Implementace webové aplikace....25
6. Testování webové aplikace....39
7. Závěr....40
Reference....42
Příloha A – obsah přiloženého CD....44
Příloha B – navržený výměnný datový formát....45

Seznam zkratek

6PE	IPv6 Provider Edge
AfriNIC	The African Network Information Centre
APNIC	The Asia-Pacific Network Information Centre
ARIN	American Registry for Internet Numbers
ATM	Asynchronous Transfer Mode
CIDR	Classless InterDomain Routing
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
IP	Internet Protocol
ISP	Internet Service Provider (poskytovatel pro připojení k internetu)
JSON	JavaScript Object Notation
LACNIC	Latin American and Caribbean Internet Address Registry
LIR	Local Internet Registry
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
MVC	Model View Controller
NAT	Network Address Translation
NTP	Network Time Protocol
RIPE NCC	Réseaux IP Européens Network Coordination Centre
RIR	Regional Internet Registry
SLAAC	Stateless address autoconfiguration
UML	Unified Modeling Language
WINS	Windows Internet Naming Service
XML	Extensible Markup Language

1 Úvod

Cílem této diplomové práce je vytvoření aplikace pro správu adresního prostoru IPv6, která jej umožní přidělovat adresy hierarchicky v různých úrovních různým vlastníkům a také výsledek vizualizovat.

Smyslem této aplikace je zjednodušit správu IPv6 uvnitř organizace prostřednictvím přehledné vizualizace, díky které administrátor uvidí různě přidělené bloky adres různým subjektům.

Důvodem použití webových technologií je zejména co největší přenositelnost na straně klienta, jednou napsaná aplikace tak může z klientského hlediska běžet na nejrůznějších zařízeních, jako jsou mobily, tablety, ale i desktopy a pracovní stanice.

Dalším faktorem je, že s aplikací budou pracovat různí uživatelé a bude pro to potřeba řídit, co který uživatel smí vidět, ať již z právních (například ochrana osobních údajů, různá obchodní a jiná tajemství), nebo různých jiných interních důvodů. Pro výměnu dat jsem již v rámci magisterského semestrálního projektu navrhl výměnný datový formát na bázi XML, který hodlám zde využít.

Jaké má aplikace ambice? Moje aplikace rozhodně nemůže pokrýt celý adresní prostor IPv6, zejména z důvodu celkového množství dat. Aplikace má ambice pomoci administrátorovi ISP při správě svého adresního prostoru a rozdělení svých adres mezi svou vlastní infrastrukturu a své zákazníky. Aplikace rozhodně může pomoci v evidenci již přidělených bloků adres a také vyhradit určité bloky adres pro expanzi. A co je u evidence nejdůležitější, je získat přehledný výstup, ve kterém se bude snadno orientovat a administrátor tak snadno nalezne vhodný volný blok adres pro nového zákazníka. O tom, co přesněji aplikace má umět, se ale zmíním dále, v sekci návrhu.

Co cílem této aplikace není: Není to globální registr všech adres z důvodu, který jsem popsal výše. Není to registrační robot, který nastavuje síťové prvky a podává žádosti o adresní prostory. Úkolem této aplikace je pouze přehledně a jasně zobrazit stav přidělení jednotlivých bloků adres v rámci organizace, případně ISP. Je to nástroj, který má zkrátka administrátorovi zjednodušit práci.

2 Co je IPv6?

IPv6 neboli IP verze 6 je protokol, který zabezpečuje přenos paketů v internetu a zajišťuje adresování jednotlivých zařízení. Je nástupcem současného protokolu IP (rovněž označován jako IPv4), který dnes plní stejnou funkci (a přes něj se realizuje většina přenosů), ale IPv6 řeší jeho vážné problémy, čímž je zejména značný nedostatek IPv4 adres. [30]

2.1 Proč IPv6?

Jak jsem již uvedl, hlavní výhodou IPv6 je množství adres k dispozici. Adresy v IPv4 mají délku 32 bitů, zapisují se jako čtveřice čísel 0-255 oddělenými mezi sebou tečkami, např.: 147.230.152.63. Adresy IPv6 mají délku 128 bitů a zapisují se jako osm čtveřic šestnáctkových číslic (0-f) mezi sebou oddělenými dvojtečkami, přičemž se vynechávají úvodní nuly v každé čtveřici a jedna (nejdelší se vyskytující) posloupnost nulových čtveřic se vynechává a zapisuje se jako dvě po sobě jdoucí dvojtečky, např.: fe80::226a:8aff:fea4:7b35. Z tohoto vyplývá maximální teoretické množství adres v každé verzi (část adres má totiž speciální význam a nelze je normálně použít). IPv4 adres je něco přes čtyři miliardy, což je méně než obyvatel na Zemi, kdežto IPv6 adres je mnohem víc (10^{38}). [1]

Zatímco u IPv4 jsme rádi za každou adresu, kterou máme, a proto musíme celý adresní prostor co nejefektivněji využít, v případě jejich nedostatku jsme nuceni k použití věcí jako je NAT, který zase má spousty jiných nevýhod, tak u IPv6 jsme schopni adresy přiřazovat strukturovaně, odpovídající hierarchické struktuře místa a prostředí, kde jsou nasazovány, např.: organizace → pracoviště → oddělení nebo organizace → budova → místnost, což může správci značně ulehčit práci v identifikaci zdroje problému v síti. [30]

2.2 Historie IPv6

Samotný počátek úvah o IPv6 začal již na počátku 90. let [2, 3], kdy nastal internetový boom. Jednou z motivací byl právě strach o nedostatek adres, zvláště v silně zalidněných regionech, kde k rozvoji došlo později (Čína, Indie). Původní odhad byl, že IPv4 adresy dojdou v letech 2002 až 2003. Nicméně se tento katastrofický scénář podařilo odvrátit za pomocí několika triků: CIDR (Classless InterDomain Routing) [10], což umožnilo vytvářet různě velké bloky dle potřeby, ale způsobilo závislost IP adresy na konkrétním poskytovateli Internetu, a NAT (Network Address Translation) [11], což umožnilo schovat více klientů za jednu veřejnou IP adresu, ale znesnadňuje vzájemnou komunikaci klientů, kteří jsou za různými NATy.

První návrh standardu vyšel v roce 1995 (The Recommendation for the IP Next Generation Protocol, http://tools.ietf.org/html/rfc1752), specifikace IPv6 protokolu v roce 1998 (Internet Protocol, Version 6 (IPv6) Specification, http://www.ietf.org/rfc/rfc2460.txt), DHCP pro IPv6 v roce 2003 (Dynamic Host Configuration Protocol for IPv6 (DHCPv6), http://www.ietf.org/rfc/rfc3315.txt). Bezestavové DHCP v roce 2004 (Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6, http://tools.ietf.org/search/rfc3736).

Podpora IPv6 v operačních systémech se vyvíjela následovně: V Linuxu je podpora od roku 1998 (jádro 2.1.8), ale experimentální statut opustila až v roce 2005 (jádro 2.6.12). Ve světě

MS Windows se podpora poprvé objevila v roce 2002 ve Windows XP Service Pack 1, v serverové řadě byly první Windows Server 2003. Ve Windows Vista a Server 2008 byla tato podpora znatelně vylepšena. [1]

V síti CESNET, do níž je připojena i Technická univerzita v Liberci, se začal protokol IPv6 testovat v roce 2000, ze začátku šlo o testování implementací protokolu v operačních systémech., zpočátku se jednalo o síť spojenou tunely (tzn. bylo několik „ostrovů“, kde šlo komunikovat přes IPv6, ale komunikace mezi nimi znamenala zabalit paket, poslat na druhý „ostrov“ přes IPv4 a pak zase ho rozbalit a poslat k cíli v rámci druhého „ostrova“ přes IPv6). Routery byly převážně PC s Linuxem, případně s NetBSD, což nebyl ideální stav v ohledu na výkon sítě. Později síť přešla na virtuální kanály ATM a od roku 2004 je IPv6 poskytováno jako běžná služba, rovnocenná s IPv4. Implementace je založena na MPLS a 6PE [4, 12, 30].

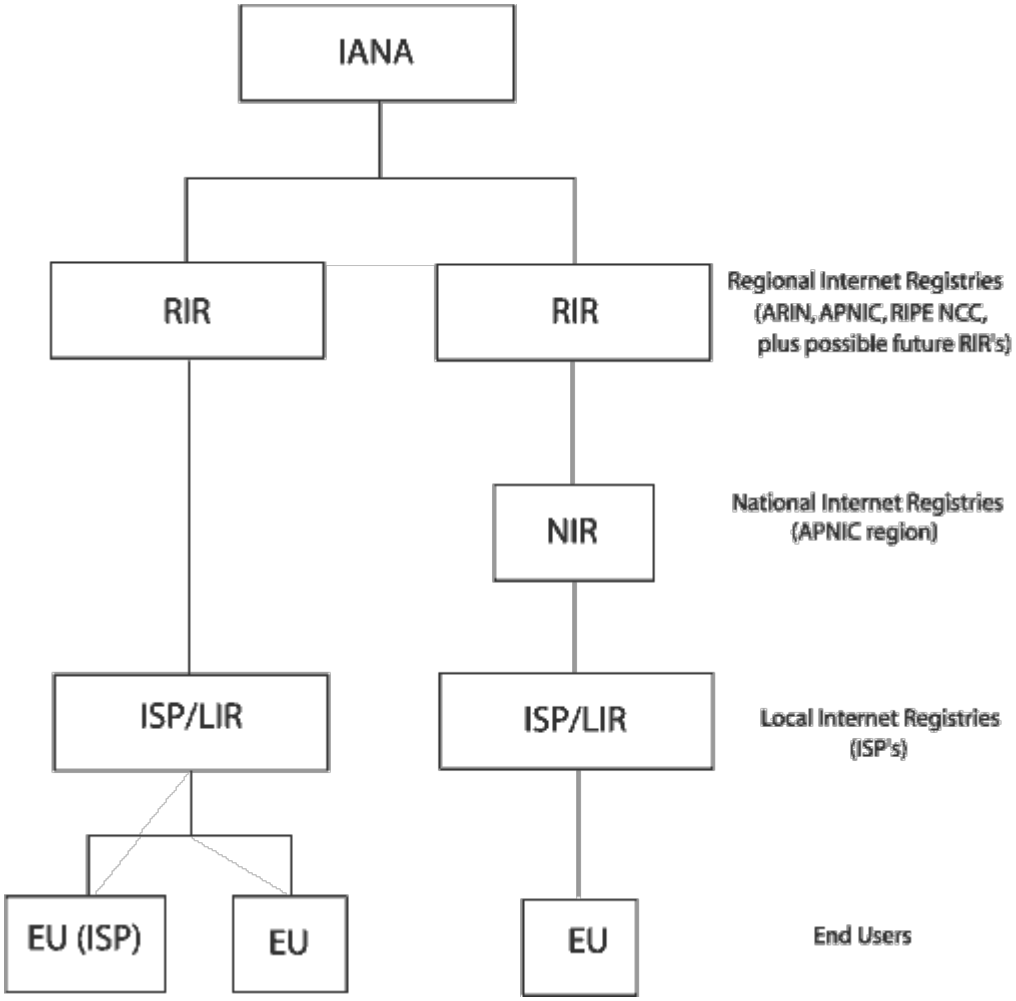
2.3 Typy IPv6 adres

Adresy se dělí do tří základních kategorií: individuální adresy (unicast), skupinové adresy (multicast) a výběrové adresy (anycast). Individuální adresy označují jedno síťové rozhraní, paket s danou cílovou adresou přijde dorazí ke zpracování právě tam a nikam jinam. Skupinové adresy označují skupinu síťových rozhraní, paket s danou cílovou adresou přijde ke zpracování ke všem rozhraním v dané skupině. Výběrové adresy rovněž označují skupinu adres, ale paket s danou cílovou adresou přijde ke zpracování na jedno rozhraní ze skupiny. Funkcionalitu všesměrových (broadcast) IPv4 adres zastávají v IPv6 skupinové. [1]

V IPv6 existují kromě globálních adres také lokální linkové (link local). Výhodou těchto adres je, že si je schopen každý stroj přidělit sám. Tyto adresy jsou jednoznačné pouze v rámci linky (jednoho Ethernetu, jedné Wi-Fi buňky) a slouží pro komunikaci po lince. Tyto adresy bývají ve směřovacích tabulkách. Důsledkem toho je, že každé rozhraní má obvykle dvě IPv6 adresy a to globální a lokální linkovou. [1, 30]

2.4 Systém přidělování adres

Systém přidělování globálních individuálních adres je nutný pro zajištění jednoznačnosti globální adresy. Nejvyšší autoritou v oblasti přidělování adres (jak IPv4, tak IPv6) je ICANN (Internet Corporation for Assigned Names and Numbers, https://www.icann.org/) [5], které má pro tuto činnost své oddělení IANA (Internet Assigned Numbers Authority, https://www.iana.org/) [6], které spravuje tzv. centrální registr adres, který zabraňuje tomu, aby se stejná adresa použila dvakrát nebo vícekrát. Z tohoto centrálního registru se adresy poskytují jednotlivým regionálním registrům (Regional Internet Registry, dále jen RIR) [6]. Těchto RIR je v současnosti na světě pět a každý spravuje určitou část světa. ARIN (American Registry for Internet Numbers) spravuje USA, Kanadu a část Karibiku, RIPE NCC (Réseaux IP Européens Network Coordination Centre) spravuje Evropu, střední východ a centrální Asii, jedná se taktéž o nejstarší RIR, APNIC (The Asia-Pacific Network Information Centre) se stará o zbytek Asie, Austrálii a Tichomoří, LACNIC (Latin American and Caribbean Internet Address Registry) spravuje oblast Latinské Ameriky a AfriNIC (The African Network Information Centre) spravuje Afriku. Tyto RIR přidělují adresy lokálním registrům (Local Internet Registry, dále jen LIR) [7, 30].



Obrázek 1: Schéma hierarchické struktury přidělování adres (https://www.ripe.net/publications/docs/ripe-641)

LIR jsou obvykle poskytovatelé připojení k Internetu a sami jsou členy RIR. LIR svoje adresy pak přidělují svým zákazníkům. Adresy se nepřidělují jednotlivě, ale v tzv blocích. Blok adres tvoří všechny adresy, které mají určitý počet prvních bitů stejný (tzv. prefix). V IPv4 se dříve obvykle volila délka prefixu 8, 16 nebo 24 bitů (dnes to již neplatí kvůli nedostatku adres), což odpovídá jednotlivým číslům v dekadickém zápisu IPv4 adresy. Blok se zapisuje pomocí lomítka za adresou, například 8.0.0.0/8 značí blok všech adres, které mají jako první číslo v adrese 8. Obdobně v IPv6 blok 2001::/16 značí všechny adresy začínající čtveřicí číslic 2001. LIR rovněž může svůj prefix dále strukturovat, například podle své vnitřní topologie (například CESNET toto dělá podle svých uzlů a pod nimi jsou bloky členů připojených v daném uzlu [14]), a to i v několika úrovních.

Nepřidělují se však všechny adresy, některé mají zvláštní význam. Takové adresy se například používají pro adresování uvnitř NAT, localhost nebo k hromadnému šíření multimediálního obsahu (multicast). Tyto zvláštní adresy jsou jak v IPv4, tak IPv6. Jejich podrobný seznam viz tabulka 1 a 2.

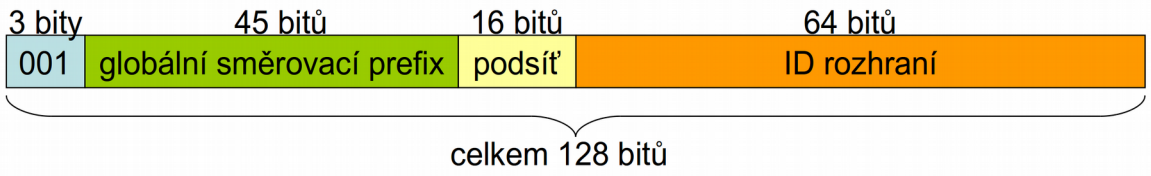
Blok adres přidělený jednomu koncovému zákazníkovi tvoří jednu síť. V IPv4 nejnižší adresa v daném bloku je adresou sítě (např. pro IPv4 blok 8.0.0.0/8 je jí 8.0.0.0), nejvyšší (8.255.255.255) je tzv. broadcast adresa, tedy cokoli přijde na tuto adresu, je rozesláno do celé

sítě. Zbytek adres je možno libovolně použít pro jednotlivé hostitele v síti. Lze také blok adres rozdělit na jednotlivé podsítě, například když poskytovatel připojení přiděluje adresy svým zákazníkům. Délka prefixu v IPv4 je libovolná.

Tabulka 1: Speciální adresy pro IPv4 (http://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml).

Address Block	Name	RFC	Allocation Date	Termination Date	Source	Destination	Forwardable	Global	Reserved-by-Protocol
0.0.0.0/8	"This host on this network"	[RFC1122], section 3.2.1.3	1981-09	N/A	True	False	False	False	True
10.0.0.0/8	Private-Use	[RFC1918]	1996-02	N/A	True	True	True	False	False
100.64.0.0/10	Shared Address Space	[RFC6598]	2012-04	N/A	True	True	True	False	False
127.0.0.0/8	Loopback	[RFC1122], section 3.2.1.3	1981-09	N/A	False[1]	False[1]	False[1]	False[1]	True
169.254.0.0/16	Link Local	[RFC3927]	2005-05	N/A	True	True	False	False	True
172.16.0.0/12	Private-Use	[RFC1918]	1996-02	N/A	True	True	True	False	False
192.0.0.0/24[2]	IETF Protocol Assignments	[RFC6890], section 2.1	2010-01	N/A	False	False	False	False	False
192.0.0.0/29	IPv4 Service Continuity Prefix	[RFC7335]	2011-06	N/A	True	True	True	False	False
192.0.0.8/32	IPv4 dummy address	[RFC-ietf-softwire-4rd-10]	2015-03	N/A	True	False	False	False	False
192.0.0.170/32, 192.0.0.171/32	NAT64/DNS64 Discovery	[RFC7050], section 2.2	2013-02	N/A	False	False	False	False	True
192.0.2.0/24	Documentation (TEST-NET-1)	[RFC5737]	2010-01	N/A	False	False	False	False	False
192.31.196.0/24	AS112-v4	[RFC-ietf-dnsop-as112-dname-06]	2014-12	N/A	True	True	True	True	False
192.52.193.0/24	AMT	[RFC7450]	2014-12	N/A	True	True	True	True	False
192.88.99.0/24	Deprecated (6to4 Relay Anycast)	[RFC-ietf-v6ops-6to4-to-historic-11]	2001-06	2015-03					
192.168.0.0/16	Private-Use	[RFC1918]	1996-02	N/A	True	True	True	False	False
192.175.48.0/24	Direct Delegation AS112 Service	[RFC-ietf-dnsop-rfc6304bis-06]	1996-01	N/A	True	True	True	True	False
198.18.0.0/15	Benchmarking	[RFC2544]	1999-03	N/A	True	True	True	False	False
198.51.100.0/24	Documentation (TEST-NET-2)	[RFC5737]	2010-01	N/A	False	False	False	False	False
203.0.113.0/24	Documentation (TEST-NET-3)	[RFC5737]	2010-01	N/A	False	False	False	False	False
240.0.0.0/4	Reserved	[RFC1112], section 4	1989-08	N/A	False	False	False	False	True
255.255.255.255/32	Limited Broadcast	[RFC919], section 7	1984-10	N/A	False	True	False	False	False

V IPv6 je posledních 64 bitů adresy (celá druhá polovina) vždy určeno pro adresu koncového zařízení v síti. Zde již nelze dělat podsítě. Toto omezení je dáno specifikací a některá zařízení



Obrázek 2: Schéma IPv6 adresy (http://www.kvd.zcu.cz/cz/materialy/9PSDS/IPv6-prezentace-04-02-10.ppt)

ani pracovat s menšími bloky adres neumí. Zbylá část adresy lze použít pro adresu sítě a podsítě. Důsledkem toho je, že nejmenší možná podsíť má adresní prostor více než čtyřmiliardkrát větší, než je celý adresní prostor IPv4. Nicméně všechny v současnosti přidělené bloky pro globální adresy začínají bity 001, což odpovídá první číslici v adrese jako 2 nebo 3. [30]

Tabulka 2: Speciální adresy pro IPv6 (http://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml).

Address Block	Name	RFC	Allocation Date	Termination Date	Source	Destination	Forwardable	Global	Reserved-by-Protocol
::1/128	Loopback Address	[RFC4291]	2006-02	N/A	False	False	False	False	True
::/128	Unspecified Address	[RFC4291]	2006-02	N/A	True	False	False	False	True
::ffff:0:0/96	IPv4-mapped Address	[RFC4291]	2006-02	N/A	False	False	False	False	True
64:ff9b::/96	IPv4-IPv6 Translat.	[RFC6052]	2010-10	N/A	True	True	True	True	False
100::/64	Discard-Only Address Block	[RFC6666]	2012-06	N/A	True	True	True	False	False
2001::/23	IETF Protocol Assignments	[RFC2928]	2000-09	N/A	False[1]	False[1]	False[1]	False[1]	False
2001::/32	TEREDO	[RFC4380]	2006-01	N/A	True	True	True	False	False
2001:2::/48	Benchmarking	[RFC5180]	2008-04	N/A	True	True	True	False	False
2001:3::/32	AMT	[RFC7450]	2014-12	N/A	True	True	True	True	False
2001:4:112::/48	AS112-v6	[RFC-ietf-dnsop-as112-dname-06]	2014-12	N/A	True	True	True	True	False
2001:db8::/32	Documentation	[RFC3849]	2004-07	N/A	False	False	False	False	False
2001:10::/28	Deprecated (previously ORCHID)	[RFC4843]	2007-03	2014-03					
2001:20::/28	ORCHIDv2	[RFC7343]	2014-07	N/A	True	True	True	True	False
2002::/16[2]	6to4	[RFC3056]	2001-02	N/A	True	True	True	N/A[2]	False
2620:4f:8000::/48	Direct Delegation AS112 Service	[RFC-ietf-dnsop-rfc6304bis-06]	2011-05	N/A	True	True	True	True	False
fc00::/7	Unique-Local	[RFC4193]	2005-10	N/A	True	True	True	False	False
fe80::/10	Linked-Scoped Unicast	[RFC4291]	2006-02	N/A	True	True	False	False	True

2.5 Přidělení IP adresy koncovému zařízení

U IPv4 jsou dvě možnosti, jakým způsobem přidělit rozhraní IP adresu, a to buď ruční konfigurací nebo pomocí DHCP (Dynamic Host Configuration Protocol). Ruční konfiguraci provádí správce sítě přímo na daném zařízení. Musí mít tedy přímo k tomuto zařízení přístup. Tato možnost je vhodná pro servery, kde chceme zajistit neměnnou IP adresu. Nevýhodou je, že každé zařízení musí správce sítě nastavit a zvolit adresy, aby nedocházelo k duplicitám. Druhou možností je DHCP, kde se adresa nastavuje automaticky za pomoci DHCP serveru. Protokol funguje tak, že klient pošle broadcast paket DISCOVER pro vyhledání DHCP serveru. Router tento DISCOVER paket zachytí a pošle správnému DHCP serveru. DHCP server zvolí vhodnou IP adresu z rozsahu, který nastavil správce sítě, a klientovi ji nabídne pomocí OFFER paketu společně s adresami některých dalších služeb jako DNS, WINS a NTP, server tuto adresu si zarezervuje. Klient odpoví paketem REQUEST, čímž nabídku přijme. Nakonec server pošle ACK paket, kterým stvrdí klientovu IP adresu a sdělí, do kdy ji má klient pronajatou. Po uplynutí této doby si musí klient IP adresu nechat obnovit. [13]

Tabulka 3: Seznam IPv6 bloků adres přidělených jednotlivým RIR (http://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml)

Prefix	Designation	Date	WHOIS	RDAP	Status	Note
2001:0000::/23	IANA	1999-07-01	whois.iana.org		ALLOCATED	2001:0000::/23 is reserved for IETF Protocol Assignments [RFC2928]. 2001:0000::/32 is reserved for TEREDO [RFC4380]. 2001:0002::/48 is reserved for Benchmarking [RFC5180]. 2001:10::/28 is reserved for ORCHID [RFC4843]. For complete registration details, see [IANA registry *iana-ipv6-special-registry*].
2001:0200::/23	APNIC	1999-07-01	whois.apnic.net		ALLOCATED	
2001:0400::/23	ARIN	1999-07-01	whois.arin.net		ALLOCATED	
2001:0600::/23	RIPE NCC	1999-07-01	whois.ripe.net		ALLOCATED	
2001:0800::/23	RIPE NCC	2002-05-02	whois.ripe.net		ALLOCATED	
2001:0a00::/23	RIPE NCC	2002-11-02	whois.ripe.net		ALLOCATED	
2001:0c00::/23	APNIC	2002-05-02	whois.apnic.net		ALLOCATED	2001:db8::/32 reserved for Documentation [RFC3849]. For complete registration details, see [IANA registry *iana-ipv6-special-registry*].
2001:0e00::/23	APNIC	2003-01-01	whois.apnic.net		ALLOCATED	
2001:1200::/23	LACNIC	2002-11-01	whois.lacnic.net		ALLOCATED	
2001:1400::/23	RIPE NCC	2003-02-01	whois.ripe.net		ALLOCATED	
2001:1600::/23	RIPE NCC	2003-07-01	whois.ripe.net		ALLOCATED	
2001:1800::/23	ARIN	2003-04-01	whois.arin.net		ALLOCATED	
2001:1a00::/23	RIPE NCC	2004-01-01	whois.ripe.net		ALLOCATED	
2001:1c00::/22	RIPE NCC	2004-05-04	whois.ripe.net		ALLOCATED	
2001:2000::/20	RIPE NCC	2004-05-04	whois.ripe.net		ALLOCATED	
2001:3000::/21	RIPE NCC	2004-05-04	whois.ripe.net		ALLOCATED	
2001:3800::/22	RIPE NCC	2004-05-04	whois.ripe.net		ALLOCATED	
2001:3c00::/22	IANA				RESERVED	2001:3c00::/22 is reserved for possible future allocation to the RIPE NCC.
2001:4000::/23	RIPE NCC	2004-06-11	whois.ripe.net		ALLOCATED	
2001:4200::/23	AFRINIC	2004-06-01	whois.afrinic.net		ALLOCATED	
2001:4400::/23	APNIC	2004-06-11	whois.apnic.net		ALLOCATED	
2001:4600::/23	RIPE NCC	2004-08-17	whois.ripe.net		ALLOCATED	
2001:4800::/23	ARIN	2004-08-24	whois.arin.net		ALLOCATED	
2001:4a00::/23	RIPE NCC	2004-10-15	whois.ripe.net		ALLOCATED	
2001:4c00::/23	RIPE NCC	2004-12-17	whois.ripe.net		ALLOCATED	
2001:5000::/20	RIPE NCC	2004-09-10	whois.ripe.net		ALLOCATED	
2001:8000::/19	APNIC	2004-11-30	whois.apnic.net		ALLOCATED	
2001:a000::/20	APNIC	2004-11-30	whois.apnic.net		ALLOCATED	
2001:b000::/20	APNIC	2006-03-08	whois.apnic.net		ALLOCATED	
2002:0000::/16	6to4	2001-02-01			ALLOCATED	2002::/16 is reserved for 6to4 [RFC3056]. For complete registration details, see [IANA registry *iana-ipv6-special-registry*].
2003:0000::/18	RIPE NCC	2005-01-12	whois.ripe.net		ALLOCATED	
2400:0000::/12	APNIC	2006-10-03	whois.apnic.net		ALLOCATED	2400:0000::/19 was allocated on 2005-05-20. 2400:2000::/19 was allocated on 2005-07-08. 2400:4000::/21 was allocated on 2005-08-08. 2404:0000::/23 was allocated on 2006-01-19. The more recent allocation (2006-10-03) incorporates all these previous allocations.
2600:0000::/12	ARIN	2006-10-03	whois.arin.net		ALLOCATED	2600:0000::/22, 2604:0000::/22, 2608:0000::/22 and 260c:0000::/22 were allocated on 2005-04-19. The more recent allocation (2006-10-03) incorporates all these previous allocations.
2610:0000::/23	ARIN	2005-11-17	whois.arin.net		ALLOCATED	
2620:0000::/23	ARIN	2006-09-12	whois.arin.net		ALLOCATED	
2800:0000::/12	LACNIC	2006-10-03	whois.lacnic.net		ALLOCATED	2800:0000::/23 was allocated on 2005-11-17. The more recent allocation (2006-10-03) incorporates the previous allocation.
2a00:0000::/12	RIPE NCC	2006-10-03	whois.ripe.net		ALLOCATED	2a00:0000::/21 was originally allocated on 2005-04-19. 2a01:0000::/23 was allocated on 2005-07-14. 2a01:0000::/16 (incorporating the 2a01:0000::/23) was allocated on 2005-12-15. The more recent allocation (2006-10-03) incorporates these previous allocations.
2c00:0000::/12	AFRINIC	2006-10-03	whois.afrinic.net		ALLOCATED	
2d00:0000::/8	IANA	1999-07-01			RESERVED	
2e00:0000::/7	IANA	1999-07-01			RESERVED	
3000:0000::/4	IANA	1999-07-01			RESERVED	
3ffe::/16	IANA	2008-04			RESERVED	3ffe:831f::/32 was used for Teredo in some old but widely distributed networking stacks. This usage is deprecated in favor of 2001::/32, which was allocated for the purpose in [RFC4380]. 3ffe::/16 and 5f00::/8 were used for the 6bone but were returned. [RFC5156]
5f00::/8	IANA	2008-04			RESERVED	3ffe::/16 and 5f00::/8 were used for the 6bone but were returned. [RFC5156]

U IPv6 fungují (až na malé odlišnosti) tytéž metody jako u IPv4, ale přibývá třetí možnost, a to bezstavová automatická konfigurace adres (SLAAC). Tato možnost na rozdíl od klasického DHCP nevyžaduje DHCP server. Tato metoda stojí na IPv6 mechanismu objevování sousedů. Využívá se toho, že každý router posílá do sítí, ke kterým je připojen tzv. ohlášení, kde rozesílá informace o adresách, které se v síti používají a jestli umí on sám posílat pakety ven ze sítě (je schopen sloužit jako default gateway). Tyto zprávy posílá každý router v určitém časovém intervalu a lze je vynutit pomocí výzvy (kterou může zaslat například klient, který žádá o IPv6 adresu). Z těchto informací si klient zjistí adresy používané v síti a svoji adresu si zvolí z adresy sítě za kterou si doplní 64 bitů jednoznačně odvozených od fyzické (MAC) adresy rozhraní. Poté následně ověří pomocí mechanismu objevování sousedů duplicity. [30]

2.6 Přidělování IPv6 adres v rámci RIPE NCC

Jak jsem uvedl výše, RIPE NCC je RIR působící v oblasti Evropy, tudíž přiděluje bloky adres jednotlivým LIR. Tudíž, aby organize mohla od RIPE NCC vůbec nějaké adresy přímo dostat, musím se stát LIR, který bude členem RIPE NCC. Pokud se LIR stát sama nechce (například jakožto malý ISP, který nechce platit členské poplatky v RIPE NCC), tak bloky adres může získat od nadřízeného ISP (od toho kupuje konektivitu a přeprodává ji svým zákazníkům) a pak proces a množství adres bude čistě záležet na tomto ISP.

Je-li organizace LIR, dostanu nejprve blok adres velikosti /32. Zaplní-li ho, tak expanduje postupně až na /29. Pro jakoukoliv další expanzi už pro ni RIPE NCC bude chtít vědět, na co ty adresy používá, tudíž mu bude muset předat další dokumentaci, která obsahuje například topologii její sítě. [17]

IXP (Internet Exchange Point, propojovací uzly, například NIX.CZ) dostávají přiděleno /64, pokud potřebují pouze jedinou síť a nikdy nebudou potřebovat více, jinak dostanou /48. Tyto adresy jsou přiděleny přímo od RIPE NCC nebo skrze LIR. [18]

Expandovat lze pouze tehdy, když využitelnost současného prostoru měřítkem zvaným HD-poměr (HD-Ratio) přeskočí stanovenou mez. Jeho přesný vztah vyjadřuje níže uvedený vzorec. Pro tento účel se jako alokovatelné objekty u RIPE NCC počítají bloky adres velikosti /56. Pokud využitelnost dosáhne aspoň 0,94, lze požádat o další adresy. Kolik procent bloků /56 musí být využito pro blok adres dané velikosti popisuje tabuka 4.

$$HD = \frac{\log(\text{počet alokovaných objektů})}{\log(\text{počet alkokovatelných objektů})}$$

Vzorec pro výpočet HD-poměru

Tabulka 4: Obsazenost bloku adres při HD-Ratio 0,94 vůči referenčním blokům velikosti /56 (https://www.ripe.net/publications/docs/ripe-655)

Prefix	Total /56s	/56s HD 0.94	Util %
10	70368744177664	10388121308479	14.76
11	35184372088832	5414630391777	15.39
12	17592186044416	2822283395519	16.04
13	8796093022208	1471066903609	16.72
14	4398046511104	766768439460	17.43
15	2199023255552	399664922315	18.17
16	1099511627776	208318498661	18.95
17	549755813888	108582451102	19.75
18	274877906944	56596743751	20.59
19	137438953472	29500083768	21.46
20	68719476736	15376413635	22.38
21	34359738368	8014692369	23.33
22	17179869184	4177521189	24.32
23	8589934592	2177461403	25.35
24	4294967296	1134964479	26.43
25	2147483648	591580804	27.55
26	1073741824	308351367	28.72
27	536870912	160722871	29.94
28	268435456	83774045	31.21
29	134217728	43665787	32.53
30	67108864	22760044	33.92
31	33554432	11863283	35.36
32	16777216	6183533	36.86

3 PHP 7

Pro tvorbu serverové části webových aplikací lze použít jazyk PHP. Jedná se jazyk primárně určený pro vývoj serverové části webových aplikací. Tedy veškerý kód v PHP běží na serveru a je před uživatelem skrytý a uživateli se dostává výstup v podobě HTML stránky (případně výstup může být XML, JSON v případě webové služby, nebo i třeba obrázek). S jazykem jako takovým už mám nějaké zkušenosti právě v oblasti webových aplikací. Aktuálně nejnovější verzí jazyka je verze 7, která má řadu novinek usnadňujících vývoj. Mezi novinky této verze jazyka patří zejména možnost deklarovat i neobjektové datové typy pro argumenty funkcí a také možnost definovat návratový typ. Dále byly některé běhové chyby a varování jako dělení nulou převedeny na mechanismus vyhazování a zachytávání výjimek, který byl také vylepšen tím, že byla implementována hierarchie výjimek podobná Javě. Také přibyla podpora anonymních tříd a mnoho dalších změn, které jsou víceméně syntaktické.

Možnost deklarace datových typů pomáhá zejména zabráněním nekontrolovatelnému šíření chyb programem. Jelikož je PHP dynamicky typovaný jazyk, stává se (obvykle při nějaké chybě), že proměnná je jiného datového typu, než si myslíme. Deklarací datového typu argumentu funkce zajistíme, že do funkce vstupuje parametr správného datového typu, který chceme. Zavoláme-li funkci se špatnými argumenty, dostaneme chybu TypeError, která je odchytitelná pomocí konstrukce try/catch. Tuto chybu také dostaneme, když návratová hodnota funkce neodpovídá deklarovanému datovému typu. Tyto typové kontroly ovšem probíhají pouze tehdy, jsou-li datové typy deklarovány, čímž je zabezpečena zpětná kompatibilita.

Převedení běhových chyb na mechanismus výjimek umožňuje jejich snazší programové zpracování, například můžeme dát provozovateli aplikace vědět, že došlo k chybě, nebo můžeme zajistit, aby se program z chyby zotavil.

Použití anonymních tříd umožňuje například typově bezpečné (s deklarací datových typů) funkcionální programování. V PHP 5.x bylo nutné vytvořit běžnou pojmenovanou třídu (a byla velmi omezená možnost deklarace datových typů). Použitím rozhraní a anonymních tříd lze totiž snadno vyrobit typově bezpečný callback na rozdíl od anonymních funkcí, kde nemůžeme požadovat po funkci určitou signaturu.

Mezi nové operátory patří operátor „??“, který je vhodný pro ošetření „null“ hodnot proměnných, dále operátor „<=>“, což je porovnávací operátor, který vrací buď -1, 0, 1, podle toho, jestli levý operand je menší než, roven, nebo větší než druhý operand. [27, 28]

Dle [29] se rovněž zapracovalo na výkonu aplikací napsaném v tomto jazyce díky zrychlení interpretru oproti verzím 5.x, což se odráží v počtu vyřízených požadavků za jednotku času.

4 Návrh aplikace

4.1 Požadavky na aplikaci

Než jsem začal navrhovat aplikaci, tak jsem si musel zjistit a dát dohromady požadavky, co má aplikace musí splňovat a co by bylo vhodné, aby splňovala (tzn. implementovat danou funkci až při dostatku času, ale myslet na možnosti její implementace již ve fázi návrhu) a také vlastnosti, které aplikace mít nemusí a jejichž opomenutí může usnadnit implementaci. Na tyto požadavky se budu dále v návrhu odvolávat:

1. Musí se jednat o webovou aplikaci, kterou bude uživatel používat pomocí webového prohlížeče.
2. Každý uživatel musí mít svá data, nesdílená s ostatními uživateli.
3. Musí být možno přidávat, rušit a měnit alokace.
4. Musí být možno přidávat, rušit a měnit informace o vlastnících bloků adres.
5. Aplikace musí být schopná graficky zobrazit aktuální stav (tak, jak ho uživatel zadal), a to v libovolné úrovni hierarchického pohledu.
6. Aplikace musí být schopna pracovat s bloky adres velikosti aspoň /64.
7. Aplikace musí umět data exportovat do formátu XML a také z téhož formátu importovat.
8. Aplikace by měla mít možnost spárovat informace o vlastnících z importu s těmi již v databázi.
9. Aplikace by měla uživateli umožnit mít více kontextů, ve kterých přiděluje, mění a ruší alokace bloků adres a tyto kontexty i pojmenovat, přejmenovat nebo mazat.
10. Aplikace by měla být schopna ukládat stavy (alokováno, rezervováno) alokací bloků adres.
11. Aplikace může umožnit uživateli specifikovat v rámci grafického zobrazení alokací nastavit vlastníkovi barvu, kterou bude v grafických zobrazeních vždy vyobrazen.
12. Aplikace může umožnit uživateli sloučit dva kontexty přidělování bloků adres do jednoho.
13. Aplikace by mohla zobrazovat obsazenost daného bloku adres například metrikou HD-Ratio podle RIPE NCC.
14. Aplikace nemusí být schopna pracovat s bloky menšími než /64
15. Postačuje, když server bude fungovat na jednom běžném operačním systému a běžné počítačové architektuře (například Linux nad amd64)

Tyto požadavky jsem získal v průběhu konzultace práce s vedoucím, nebo plynou již ze zadání. Bod 2 plyne z důvodu ochrany osobních údajů, neboť údaje o vlastnících bloků

obsahují údaje jako adresa, telefon a e-mail. Body 6 a 14 vyplývají z mechanismů přidělování adres v síti koncovému zařízení. Bod 8 vychází z principu deduplikace dat, tedy principu neukládání stejné informace dvakrát. Samozřejmě, tento přístup bude mít své limity, například omezení vyplývající z bodu 2, tudíž deduplikaci půjde realizovat pouze v rámci dat jednoho uživatele. Bod 9 vyplývá z toho, aby uživatel, když chce něco například vyzkoušet, nerozbíjel svoji konfiguraci (strom alokací) a nemusel k tomuto účelu zakládat nový uživatelský účet. Bod 10 vyplývá z možnosti vyhrazení prostoru pro expanzi a zabránění tím fragmentaci adresního prostoru a usnadnění směrování. Bod 11 je z důvodu usnadnění orientace v grafickém pohledu, aby bylo lépe vidět jednoho vlastníka v různých úrovních hierarchického pohledu nebo znázornit vlastníka barvou, kterou se obvykle prezentuje (což je běžné například u politických stran). Bod 12 zase reprezentuje uživatelský komfort. Bod 13 by ocenili LIR, kteří jsou členy RIPE NCC, jelikož by viděli, jestli mohou expandovat nebo ne. Bod 15 říká, že serverová část nemusí být schopna běžet na mnoha různých softwarových a hardwarových platformách, ostatně to ani není v mých silách na nich aplikaci otestovat.

4.2 Volba technologie

Důležitou fází návrhu aplikace je volba technologie, neboť každá technologie má svoje silné a slabé stránky, což může do velké míry ovlivňovat výsledný model.

Jelikož aplikace musí ukládat data a v nich vyhledávat a filtrovat a také je měnit a mazat, rozhodl jsem se, že pro tento účel použiji relační databázi. Vybral jsem open source databázi MariaDB, což je fork jiné open source, velice populární databáze MySQL. Výhodou MariaDB je jednak kompatibilita s MySQL, ale zejména to, že je na rozdíl od MySQL pořád ve velkém vyvíjena, kdežto u MySQL se vývoj vázne od akvizice Sun Microsystems Oraclem. Tento fakt je víceméně podpořen také tím, že mnohé webhostingy nahradily právě MySQL databází MariaDB, což také vypovídá o kvalitě MariaDB.

Dále jsem musel zvolit programovací jazyk, v němž budu implementovat serverovou část aplikace. Programovací jazyk musí mít podporu pro psaní webových aplikací, respektive jejich serverové části. Toto lze realizovat buď jazykem přímo určeným pro psaní webových aplikací, nebo jazykem obsahujícím vhodnou knihovnu nebo framework. Dále by bylo velice vhodné, aby jazyk podporoval objektově orientované programování. Zvolil jsem jazyk PHP, neboť s ním mám pro tvorbu webových aplikací už určité zkušenosti a je přímo určený pro problematiku serverové části webových aplikací. Zvolil jsem aktuálně poslední stabilní verzi (7.0). Tato verze obsahuje řadu novinek, které zejména zamezují nekontrolovatelnému šíření chyb programem, což velmi usnadňuje vývoj aplikace. PHP ale ovšem řeší pouze problém backendu aplikace, pro frontend je třeba zvolit nějaké další řešení.

Pro webové aplikace je obvyklé, že je klientem webový prohlížeč, takže volba technologie frontendu je závislá na výčtu technologií, se kterými je schopen webový prohlížeč pracovat. Logickou volbou je kombinace HTML, CSS a JavaScriptu. Nad touto kombinací je celá řada frameworků usnadňujících vývoj této části aplikace, například Bootstrap [19] a Foundation [20]. Cílem těchto frameworků je nabídka již hotových prvků pro web s přijatelným designem, jako například menu, formuláře, kontejnery, vzhled stránkování a jiné.

Pro svoji práci jsem si vybral framework Bootstrap, protože už s ním mám aspoň nějaké zkušenosti. Bohužel, tyto frameworky nepokryjí zcela všechny myslitelné funkce (například někdy je třeba napsat, co má tlačítko dělat), takže je i někdy potřeba sáhnout po JavaScriptu. Pro usnadnění vývoje těchto součástí aplikace jsem sáhnul po knihovně jQuery, kterou tak jako tak musím požít, protože už ji využívá mnou zvolený framework Bootstrap.

Pro další usnadnění (zejména vyhnutí se chybám) využívám ve svém JavaScriptovém kódu syntaxi podle standardu ECMAScript verze 6. Tento standard umožňuje použít syntaxi se sémantikou obdobnou ostatním objektově orientovaným jazykům. Tohoto standardu už jsem využil při psaní jedné aplikace a mám s ním velice příjemné zkušenosti. Tyto zkušenosti pramení zejména z faktu, že nedocházelo k častým chybám, kterých jsem se dopouštěl při vývoji jiné předchozí aplikace pod standardem ECMAScript verze 5. Sémantika konstrukcí ve standardu ECMAScript verze 5 v určitých případech je zde diametrálně odlišná například od jazyka Java, se kterým mám mnohem větší zkušenosti, než s JavaScriptem, ale na vývoj frontendu webové aplikace se prostě nehodí. [21]

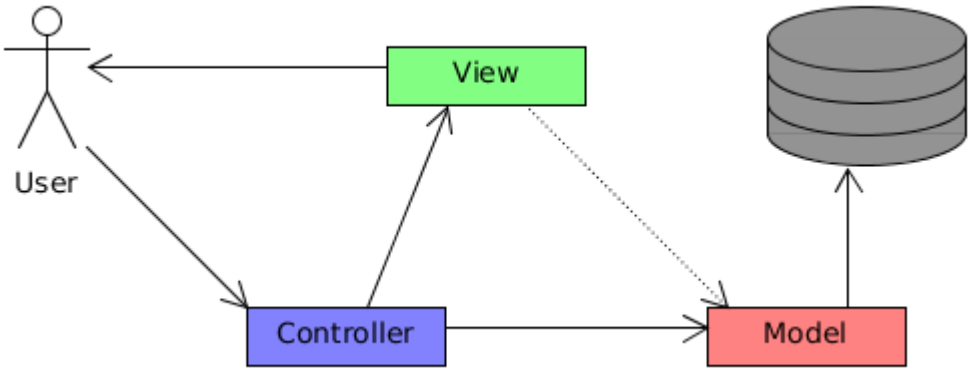
ECMAScript verze 6 toto řeší tím, že vedle možností, které fungují v ECMAScriptu verze 5, nabízí i syntaktické možnosti, které více odpovídají sémantice například Javy. Mezi takové možnosti patří například deklarace proměnných pomocí klíčového slova „let“, kdy má proměnná platnost v rámci bloku (tzv. block scope). Dále to jsou tzv. Arrow Functions, které usnadňují tvorbu hojně používaných callbacků hlavně díky intuitivnější práci s referencí „this“ a nechybí možnost pracovat se třídami objektů pomocí klíčového slova „class“, což je přístup běžně používaný v ostatních objektově orientovaných programovacích jazycích. Mezi nevýhody patří podpora ve webových prohlížečích, jelikož tato funkcionalita byla do nich mnohdy dodána až nedávno, podle [22] například Arrow Functions neumí dosud poslední verze prohlížeče Internet Explorer z dílny Microsoftu a ještě hůře je na tom prohlížeč Safari od Applu, který neumí ani klíčové slovo „let“. Tento problém se dá eliminovat za použití transpileru Babel [23], který převede kód odpovídající ECMAScript verze 6 na ECMAScript verze 5. Nicméně použití transpileru znesnadňuje ladění přímo v prohlížeči, ale toto lze obejít laděním v prohlížeči, který JavaScript podle standardu ECMAScript verze 6 umí, což jsou současné verze Firefoxu a Chromu, případně existuje vývojářská verze Firefoxu, Firefox Developer Edition [24], což je verze Firefoxu založená na aplha verzi (aktualizační kanál Aurora) Firefoxu, která obsahuje podporu pro více nových technologií, určených pro testování. Firefox Developer Edition je zvláštní variantou prohlížeče, určenou vývojářům webových aplikací, a proto je to vhodné prostředí pro testování klientské části aplikace.

4.3 Architektura aplikace

Pro webovou aplikaci je typický model komunikace typu klient – server. Tento osvědčený model spočívá v tom, že klient vyšle požadavek (autentizace, zobrazit určitá data, přidat, změnit nebo smazat záznam) a server odešle odpověď (informace o úspěchu, zobrazení požadovaných dat). Je-li klientem webový prohlížeč, tak obvykle odpovědí je HTML stránka obsahující dané informace, která se zobrazí uživateli. Jelikož očekávaným klientem je webový prohlížeč používající HTTP protokol, tak je nutno zvolit model komunikace klient – server.

Pro vlastní aplikaci jsem zvolil architekturu MVC (Model-View-Controller), kdy se aplikace skládá ze tří základních částí: Model obsahuje základní logiku a pracuje s databází. View se stará o prezentaci dat a Controller obsluhuje řízení. Dle [25] existuje hodně variací na přesnou realizaci architektury MVC.

Navrhl jsem, že do části Model zahrnu komunikaci s databází, vztahy mezi entitami, různé metody pro výpočet různých vlastností, včetně konverzí interních datových struktur na řetězec (nikoliv však obsahujících HTML) jako je konverze zadané IPv6 adresy ve standardním formátu na 64bitové číslo (jelikož postačuje schopnost práce s prvními 64 bity adresy) a naopak. Samozřejmě součástí modelu jsou také filtry na hledání a také vytváření stromů.



Obrázek 3: Architektura aplikace, šipky naznačují směry interakce.

Controller řeší požadavky uživatele, ověřuje uživatelský vstup, data z něj pošle do modelu, který je vyhodnotí (ale Controller určuje, co se bude vyhodnocovat); například pokud uživatel vyhledává, tak Controller zavolá patřičnou funkci v modelu nad vyhledávaným výrazem a tím získá relevantní výsledky a dále nad nimi může zavolat nějakou další funkci modelu, například jejich mazání, ale Controller neví nic o implementaci vyhledávání (například jestli se vleze do databáze, využije cache, nebo použije nějaká externí služba) ani o implementaci mazání (neví vůbec, jak jsou data uložena). Nakonec pošle data do View, které se postará o prezentaci dat.

View dostane data (obvykle třídy z Modelu) a vytvoří patřičný výstup (obvykle HTML). Je důležité, že View pouze data z Modelu čte, nezapisuje do něj. Tímto způsobem lze rovněž jednoduchým způsobem udělat z webové aplikace webovou službu a to tím, že pouze přepíšeme View část aplikace, aby vracela XML nebo JSON místo HTML.

4.4 SEO optimalizace

Jelikož v současné době hodně rozhodují vyhledávače o tom, jestli uživatelé na web přijdou nebo ne, byla by chyba se nezmínit o faktorech, které dělají stránky úspěšnější u vyhledávačů. V první řadě je třeba se rozhodnout, co vůbec může vyhledávač vidět, jaké stránky ano, a které jsou před vyhledávači skryty. Vyhledávač může pouze vidět to, co uvidí nepřihlášený uživatel. Jelikož uživatelé nic nesdílejí mezi sebou žádné informace (bod 2 mezi požadavky), tak je ani nemůže vidět nepřihlášený uživatel. Jelikož celá aplikace je postavena na práci s daty konkrétního uživatele, tak nepřihlášený uživatel uvidí pouze přihlašovací stránku, aby

se z něj mohl stát přihlášený uživatel a získal tak přístup ke svým datům. Z toho vyplývá, že vyhledávač uvidí pouze přihlašovací stránku, jinak nic. Vyhledávače tedy nemají, co by v této aplikaci indexovaly. Jakákoliv optimalizace pro vyhledávače je z tohoto důvodu zbytečná a proto se jí dále nebudu vůbec zabývat.

4.5 Datový model

Datový model vychází hlavně z potřeby, jaká data se budou ukládat. Jelikož musím řešit autentizaci uživatelů, musím o uživatelích ukládat následující údaje:

- uživatelské jméno, pod kterým se uživatel přihlašuje, toto jméno musí být unikátní, osobně nejsem příznivcem přihlašování pomocí e-mailové adresy, protože může být velice dlouhá a tudíž náročná pro napsání správně, pokud se uživatel chce přihlásit například z jiného počítače, kde ji nemá uloženou.
- otisk hesla, aby šlo poznat, zda uživatel zadal správné heslo při přihlášení, současně musí být zabráněno únikům hesel přímo z databáze
- uživatelův e-mail, jakožto komunikační kanál pro případné notifikace nebo i aktivaci uživatelského účtu

Každý uživatel může zakládat kontexty (bod 9), které jsem pojmenoval jako schémata. Každý uživatel může mít několik schémat. A aby se uživatel ve svých schématech neztratil, tak si je pojmenuje. Musím tedy uložit následující údaje:

- název schématu, jak si ho zvolil uživatel, musí být unikátní v rámci jednoho uživatele.
- reference na uživatele, kterému dané schéma patří a jenom on s ním může jakkoliv manipulovat

Každý uživatel ukládá informace o vlastnících bloků adres, jeden vlastník může vlastnit více bloků adres, může se vyskytovat ve vícero schématech, ale podle bodu 2 je přístupný pouze jediným uživatelem, a to tím, který ho založil. Každý vlastník bloků adres se nějak jmenuje a má nějaké kontaktní údaje. Ke každému vlastníkovi je tudíž nutno uložit následující údaje:

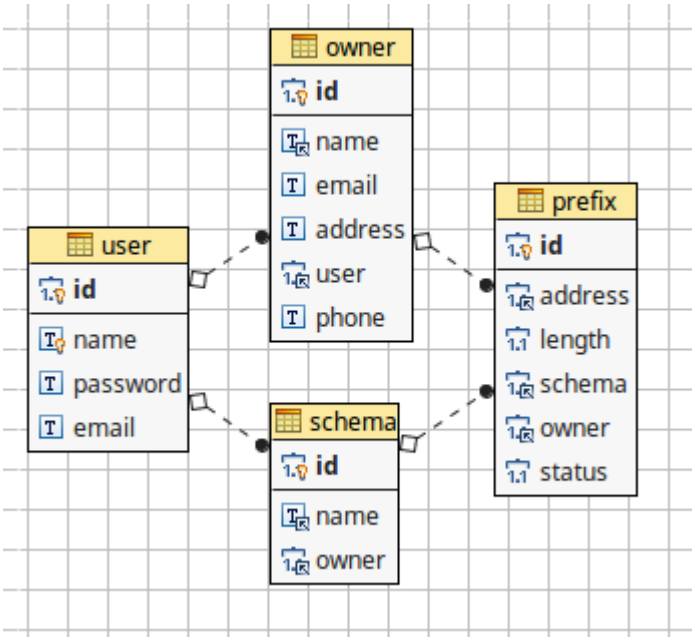
- název vlastníka (může se jednat buď o jméno fyzické osoby nebo název firmy)
- poštovní adresu
- e-mail
- telefon
- referenci na uživatele, který daného vlastníka vytvořil

Naposled tu máme informace o jednotlivých blocích adres. Tyto bloky mají svoji adresu, délku prefixu, nacházejí se v určitém schématu a jsou přiřazeny nějakému vlastníkovi. Jsou také rovněž v nějakém stavu, a to alokováno nebo rezervováno. Budu potřebovat tyto atributy:

- adresa prefixu
- délka prefixu

- reference na schéma, kde je alokace provedena
- reference na vlastníka, kterému byl daný blok adres přidělen
- status alokace, tj. jestli byl blok adres vlastníkovi alokován, nebo je pouze pro něj rezervován (například pro budoucí expanzi)

Tento datový model lze v databázi realizovat čtyřmi tabulkami, mezi kterými jsou vazby 1:n. Pro zajištění referencí jsem do každé tabulky přidal atribut „id“, který jednoznačně identifikuje záznam v tabulce a je tudíž vhodný pro zprostředkování referencí. Hodnota tohoto atributu bude automaticky generována pro zajištění unikátnosti. Výsledkem je ERA model, který jsem získal za pomocí databázového klienta DBeaver, viz obrázek 4.



Obrázek 4: Návrh databáze

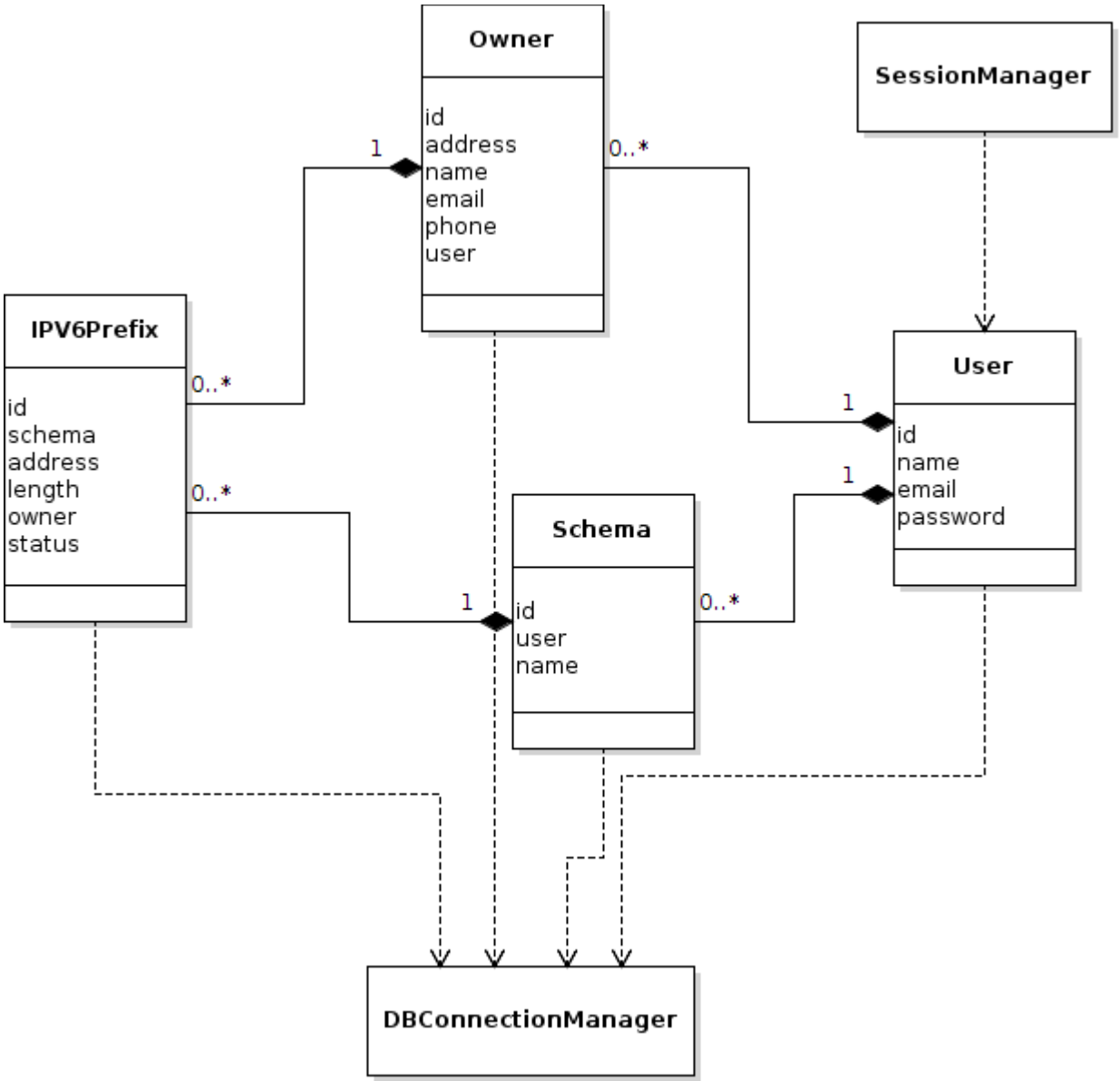
4.6 Objektový model

S datovým modelem úzce souvisí model objektový pro reprezentaci problému v objektově orientovaném programovacím jazyce, zejména pro část Model z architektury MVC. Pro každou tabulku, kterou jsem založil v databázi, musím udělat třídu, která se o ni bude starat (bude mít stejné atributy a bude schopna vytvořit instanci ze záznamu v databázi a naopak instance dané třídy musí být schopna se do databáze uložit). Dále je potřeba spravovat samotné spojení s databází, pro tento účel jsem navrhl třídu DBConnectionManager. Účelem této třídy bude z konfigurace získat přístupové údaje k databázi a připojit se k ní a toto spojení nabídnout ostatním třídám v rámci Modelu, kdykoliv budou pracovat s databází. Jako vhodné zde považuji použití návrhového vzoru Singleton.

Další věc, na kterou je třeba myslet, je zpracování sezení, tj. musím být schopen zjistit, jestli komunikuji stále se stejným uživatelem nebo už jiným a minimalizovat nebezpečí, že za uživatele se bude vydávat někdo jiný, například vystupující pod stejnou IPv4 adresou, což je velice běžná záležitost kvůli NAT. V jazyce PHP je pro ukládání údajů o sezení superglobální pole $_SESSION, nicméně nechci, aby se mohlo do něj kdykoliv bez kontroly zapisovat.

Proto jsem ho obalil třídou SessionManager, která je navržena podle návrhového vzoru Singleton (toto superglobální pole je jenom jedno a váže se na aktuální sezení) a ve zbytku aplikace superglobální pole $_SESSION nebudu využívat. SessionManager se tedy bude zabývat záležitostmi okolo přihlašování uživatelů. Výslednou situaci bych popsal UML diagramem (obrázek 4).

Tento UML diagram zachycuje návrh Modelu v rámci architektury MVC. Za zmínku stojí využití kompozice, které znamená, že bez uživatele nemůže existovat ani schéma ani vlastník bloku adres (jelikož tyto entity jsou lokální pro uživatele) a přidělený blok adres nemůže zůstat bez vlastníka a bez schématu, které mu definuje kontext. Toto jsem realizoval i na datové vrstvě pomocí cizích klíčů a kaskádovitého mazání. Tento přístup mi zajišťuje, aby v systému nezůstávala osiřelá data (tím myslím data, ke kterým už nikdo přístup mít nebude a nemají žádný systémový význam a tím pádem pouze zabírají místo na disku nebo v paměti serveru a zpomalují ho, jednalo by se o jistou formu leaku).

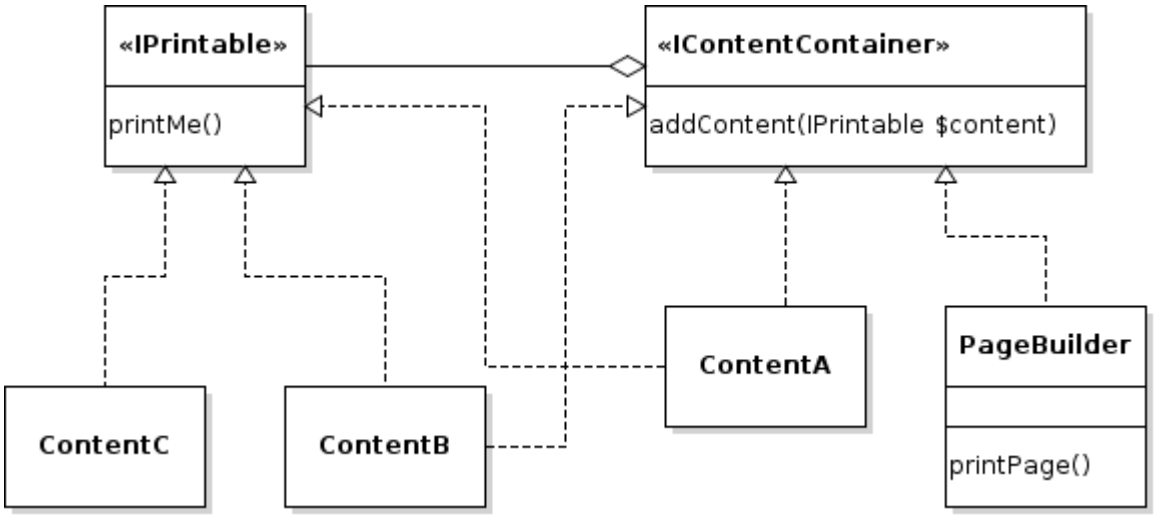


Obrázek 5: Návrh implementovaných tříd reprezentující problém

4.7 Návrh systému prezentace dat

Obvyklá struktura webové stránky je, že má hlavičku, tělo a patičku. Obsah hlavičky a patičky bývá na skoro všech stránkách jednoho webu stejný nebo aspoň podobný (skoro stejný). V hlavičce obvykle najdeme kromě technických informací v tagu „<head>“ také titulek stránky a navigaci webu, v patičce se obvykle nachází nějaká informace o copyrightu, případně nějaký kontakt na autora stránky nebo provozovatele webu. Také se do tohoto místa umisťují některé klientské skripty, aby nedocházelo ke zpomalování načítání obsahu stránky (protože skript se načte až po obsahu stránky, což může u některých javascriptových knihoven znamenat poměrně dost dat). Pro realizaci tohoto modelu jsem navrhl třídu PageBuilder, Pomocí této třídy zajistím, aby se napřed na výstup stránky zapsala hlavička, pak vlastní obsah a nakonec patička. Hlavička a patička bude vesměs vždycky stejná, o to se může postarat PageBuilder sám, obsah se mění, ale na to existuje velice úspěšný recept: polymorfismus. Díky polymorfismu PageBuilder nemusí vědět, jak přesně obsah vypadá, stačí, aby se obsah uměl vypsat.

Co se týče jednotlivých druhů obsahů, toto bude předmětem implementace aplikace, v rámci návrhu postačuje vědět, jaké mají všechny obsahové třídy rozhraní (a navíc jaké všechny obsahové třídy budu potřebovat, zjistím až během implementace). Další možností tohoto systému je možnost vytvořit obsah, který bude pouze kontejnerem pro jiný obsah (a může takových obsahů obsahovat více). Výsledek jsem zakreslil do UML diagramu (obrázek 5).



Obrázek 6: UML diagram popisující návrh architektury prezentace dat

4.8 Návrh Controlleru

Poslední částí z architektonického modelu MVC, který jsem zde v návrhu nepopsal, je část Controller, který zpracovává uživatelský vstup. Jelikož hlavní stránka (ta, kterou uživatel uvidí jako první, jinak řečeno vstupní bod do aplikace) je obsluhována souborem „index.php“ (jedná se o standardní vlastnost zvolené technologie), patří do této části aplikace. Aplikaci jsem navrhl tak, že soubor „index.php“ bude sloužit jako tzv. „master“ Controller a podle parametrů v URL předá pak řízení jinému Controlleru, který obslouží vlastní uživatelský požadavek.

4.9 Návrh vzhledu aplikace

Jako základní vzhled aplikace jsem převzal výchozí vzhled frameworku Bootstrap. Výhodou tohoto řešení je, že aplikace bude barevně a tvarově konzistentní a bude respektovat moderní požadavky a přitom bude na pracnost pokud možno co nejjednodušší. Většinu widgetů hodlám přebrat přímo z frameworku Bootstrap. Základní rozvržení stránky jsem navrhl již běžně používaný s menu nahoře a obsahem pod menu. Menu jsem zvolil jako vysouvací, ale pouze s jednou úrovní vysouvání, takže menu má celkem dvě úrovně (tlačítka + roletková menu). Tímto přístupem hodlám získat přehlednost a uživatelský komfort, jelikož pro mě jsou víceúrovňová rozbalovací menu špatná na ovládání. Nicméně možnost rozbalovacího menu dává určitou flexibilitu.

Následuje návrh grafické vizualizace. Pro ten jsem zvolil čtvercovou síť, kde každý čtverec reprezentuje jeden blok adres. Čtvercovou síť je nutno zvolit tak, aby byl počet čtverců mocninou čísla 2. Jedině tímto způsobem lze zajistit to, aby byl jeden blok adres (zvolená úroveň pohledu) plně pokryt čtverci, kde každý čtverec reprezentuje menší blok stejné velikosti. Čtyři nebo šestnáct čtverců jsem zavrhl, jelikož je to hodně málo na velikost adresního prostoru IPv6, zatímco 256 čtverců dává nutnost umístit jich do sítě 16×16, takže na běžných monitorech (o rozlišení FullHD, které je v současnosti nejběžněji používáno [26]) nelze umístit popisky a ztrácí se tak přehlednost. Zvolil jsem tedy 64 čtverců uspořádaných do sítě 8×8. Tyto čtverce jsem oindexoval v pořadí zobrazených v tabulce 5:

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

Tabulka 5: Tabulka popisující pořadí rozmístění zobrazovaných bloků adres při grafické vizualizaci

V tomto pořadí jsem umístil bloky adres do vizualizace. Cílem tohoto uspořádání je, aby související bloky tvořily pokud možno čtverec z několika menších čtverců.

4.10 Návrh výměnného datového formátu XML

Tento formát jsem již navrhoval v rámci svého semestrálního projektu [30] před vlastními pracemi na této diplomové práci. Pro definici jazyka jsem si vybral XML Schema pro jeho širokou podporu v různém softwaru, zejména v IDE. Více informací v mém magisterském semestrálním projektu [30].

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ipv6data>
  <prefixes>
    <prefix address="2001:0000::" length="23" owner="1" state="ALLOCATED"/>
    <prefix address="2001:0200::" length="23" owner="2" state="ALLOCATED"/>
    <prefix address="2001:0400::" length="23" owner="3" state="ALLOCATED"/>
    <prefix address="2001:0600::" length="23" owner="4" state="ALLOCATED"/>
    <prefix address="2001:0800::" length="23" owner="4" state="ALLOCATED"/>
    <prefix address="2001:0a00::" length="23" owner="4" state="ALLOCATED"/>
  </prefixes>
  <owners>
    <owner id="1">
      <name>IANA</name>
      <email>foo@iana.org</email>
      <phone>123456789</phone>
      <address>Foo</address>
    </owner>
    <owner id="2">
      <name>APNIC</name>
      <email>foo@iana.org</email>
      <phone>123456780</phone>
      <address>Bar</address>
    </owner>
    <owner id="3">
      <name>ARIN</name>
      <email>foo@iana.org</email>
      <phone>123456781</phone>
      <address>Baz</address>
    </owner>
    <owner id="4">
      <name>RIPE</name>
      <email>foo@iana.org</email>
      <phone>123456782</phone>
      <address>Foobar</address>
    </owner>
  </owners>
</ipv6data>
```

Ukázka výměnného datového formátu. Data jsou převzata z tabulky 3, kontaktní údaje jsou fiktivní.

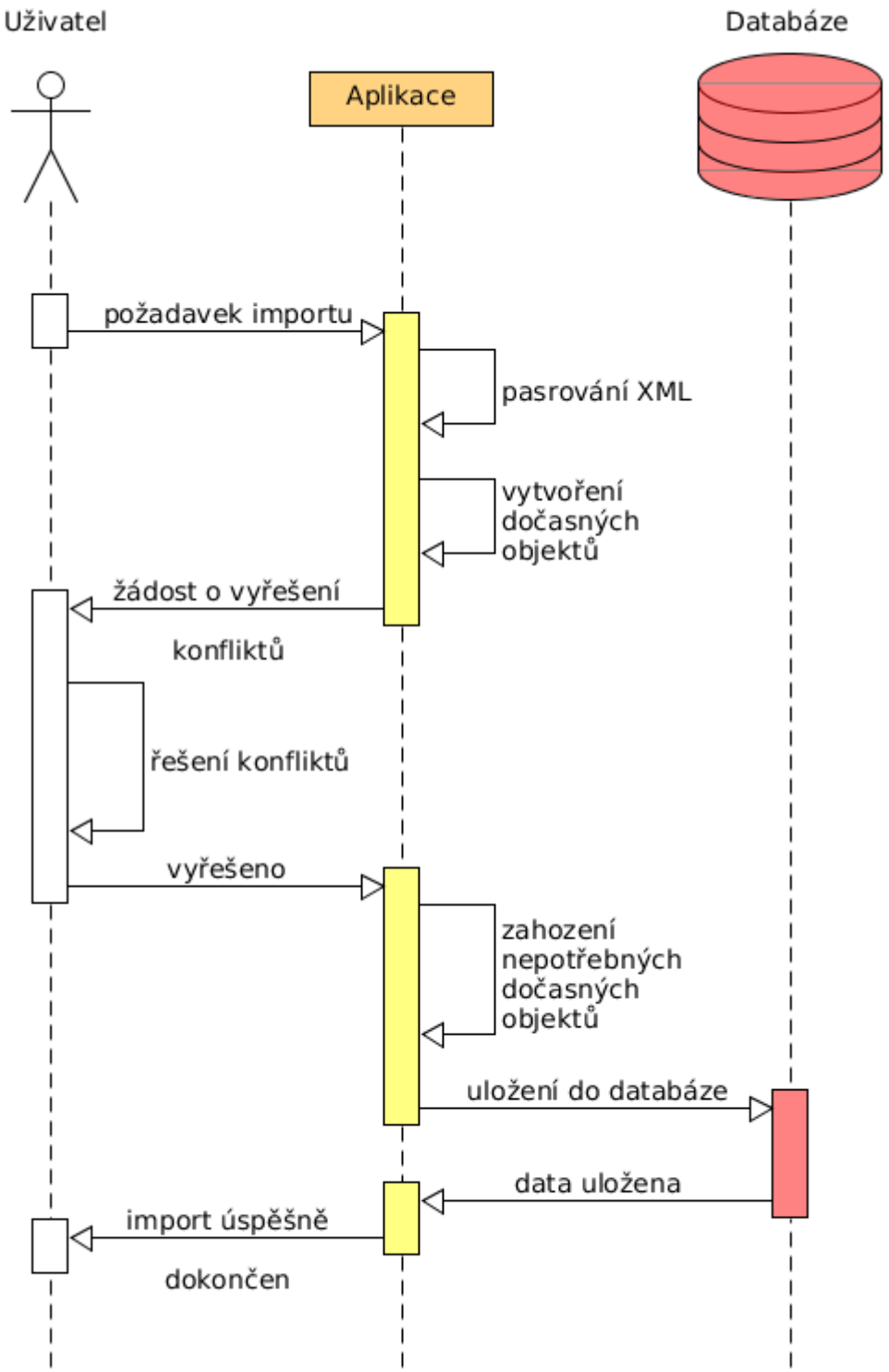
4.11 Návrh XML importu a exportu

Postup importu jsem navrhl následujícím způsobem:

1. Uživatel vybere XML soubor ze svého počítače, který chce importovat.
2. Aplikace soubor přečte, rozparsuje a vytvoří dočasné objekty reprezentující obsah souboru
3. Aplikace vyhledá konflikty s údaji, které jsou již v databázi (například podobné vlastníky jako Firma s. r. o. a Firma a. s.), a tyto konflikty předá uživateli zpět k vyhodnocení.
4. Uživatel tyto konflikty vyřeší výběrem správné hodnoty a odešle svůj výběr do aplikace.
5. Aplikace nahradí určité dočasné objekty těmi již v databázi (podle vyřešení konfliktů uživatelem).
6. Aplikace zahodí nepotřebné dočasné objekty a uloží výsledek importu do databáze.
7. Aplikace uživateli sdělí úspěch importu.

Tento postup zachycuje diagram (obrázek 6).

Postup exportu do XML jsem navrhl řešit pomocí patřičné třídy ve View. Exportovat se bude vždy pouze jedno schéma, a to s pouze vlastníky bloků adres, kteří figurují v exportovaném schématu.



Obrázek 7: Diagram popisující proces importu dat z formátu XML

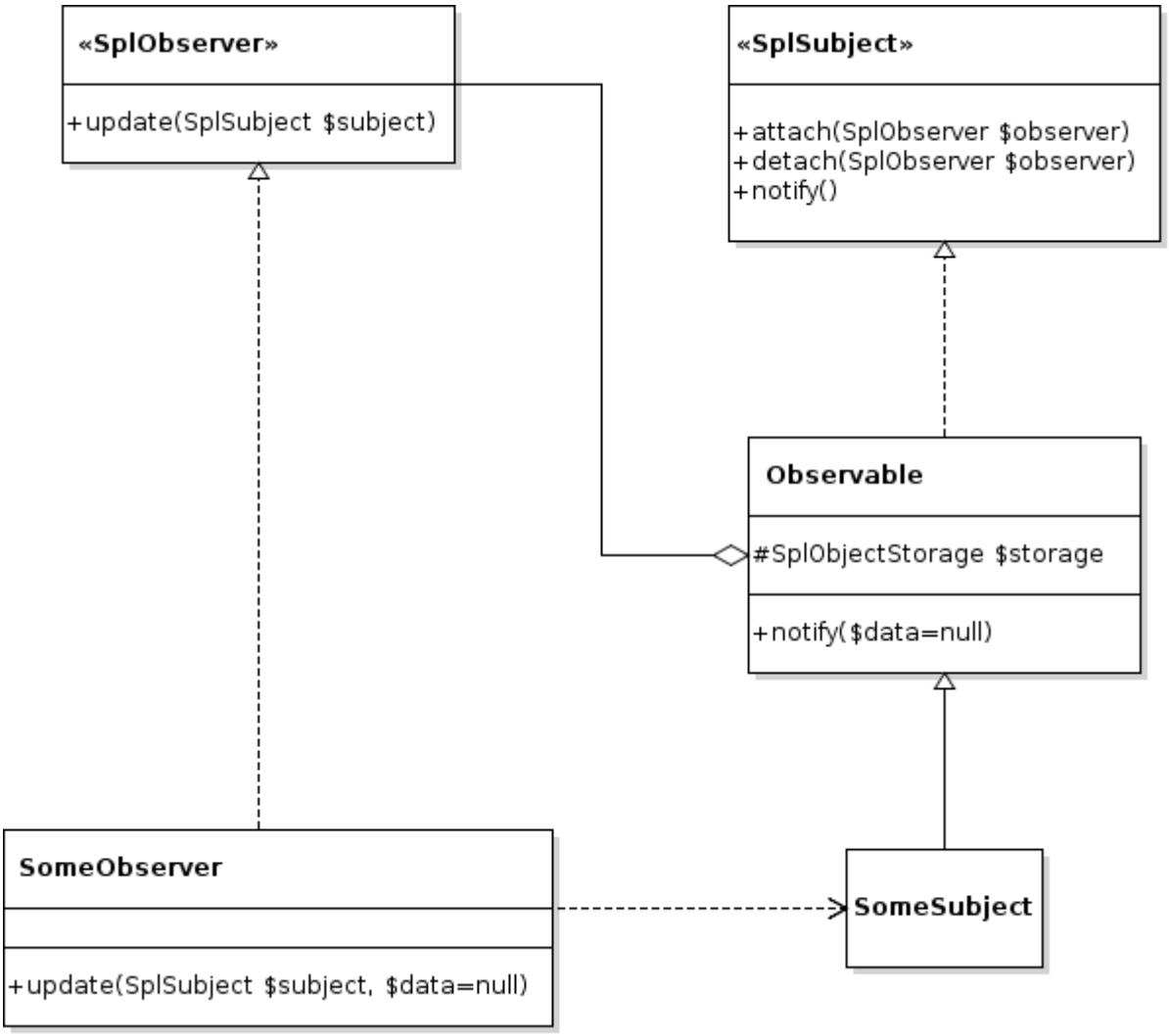
5 Implementace webové aplikace

Webovou aplikaci jsem implementoval na navržených technologiích, těmi jsou: programovací jazyk PHP verze 7.0 pro serverovou část a framework Bootstrap pro klientskou část, za použití MVC architektury.

Aplikaci jsem vyvíjel v souladu se zásadami objektově orientovaného programování. Dával jsem si záležet, abych nevytvořil nějakou cyklickou závislost.

5.1 Návrhový vzor Observer

V místech, kde byla nutná zpětná vazba, jsem použil návrhový vzor Observer. V jazyce PHP verze 7.0 je tento návrhový vzor implementován pomocí dvou rozhraní, a to SplObserver a SplSubject. Zatímco rozhraní SplObserver jsem použil přímo, nad SplSubject jsem vytvořil abstraktní třídu Observable. Abstraktní třída Observable obsahuje reference na třídy implementující SplObserver, kterým má pak zaslat zprávu. A abych mohl rozlišit typ zprávy, tak jsem musel přidat nepovinný parametr k metodám notify() a update() pro kompatibilitu definic s rozhraními SplObserver a SplSubject. Tento parametr je naplněn objektem značící typ zprávy a je pro něj užit návrhový vzor Crate. Situaci znázorňuje diagram na obrázku 7.



Obrázek 8: UML diagram popisující návrhový vzor Observer.

Takto implementovaného návrhového vzoru Observer jsem použil ve vícero případech, o kterých se zmíním dále.

5.2 Typově bezpečný přístup k atributům objektů

Jelikož PHP ve verzi 7.0 nabízí více možností na kontrolu datových typů, rozhodl jsem se této možnosti využít pokud možno co nejvíce, abych zamezil co nejdříve chybám a omezil tím jejich šíření skrz celou aplikaci. Nicméně celý systém má ještě několik omezení.

Datové typy lze deklarovat pouze u metod a funkcí, a to jak u argumentů, tak i u návratové hodnoty, u proměnných nic takového nelze dělat. Proto jsem u všech tříd až na výjimky udělal veškeré atributy privátní a pro přístup k nim ze vnějšku třídy používám metody. Například pro atribut $value používám metodu getValue() pro získání hodnoty a setValue($newVaule) pro její nastavení. Nepoužívám tedy „magické“ metody __get() a __set() běžně používané v jazyce PHP, neboť bych přišel o výhody typové kontroly a na místo toho používám způsob obvyklý v jazyce Java. Dalším problémem, na který jsem narazil, je nemožnost vrátit null, když je použita typová kontrola. Tento problém jsem obešel pomocí výjimky. Pro tento účel jsem vytvořil výjimku NoSuchElementException. Dále PHP ve verzi 7.0 nemá možnost signalizovat, že funkce nebo metoda nevrací nic (typická vlastnost setterů). Toto nelze bohužel nijak jinak vyřešit, než absencí deklarace návratového datového typu, což má bohužel tu nepříjemnou vlastnost, že přicházíme o kontrolu, že metoda skutečně nevrací nic. Elegantní řešení těchto problémů je až v návrzích pro specifikaci verze 7.1 jazyka PHP, jehož stabilní implementace dosud nebyla vydána.

Dalším omezením u datových typů argumentů a návratových hodnot je, že nelze nijak deklarovat typ hodnot uvnitř pole, lze tedy deklarovat, že funkce vrací pole nějakých hodnot. Toto omezení také hodně omezuje napovídání vývojovým prostředím. Jazyk PHP nezná generika.

5.3 Typově bezpečný výčet

Problém jazyka PHP je, že nepodporuje objekty v konstantách. Problém jsem řešil pomocí třídy, která reprezentuje daný výčtový typ a přístup k jednotlivým položkám řeším pomocí veřejných statických metod. Tímto způsobem mám zaručenou typovou kontrolu a tu vlastnost, že takto vytvořené položky již nelze přepsat žádnou jinou hodnotou.

5.4 Práce s databází

Pro práci s databází jsem využíval třídu PDO, která je přímo součástí jazyka PHP. Abych snížil režii a zajistil konzistenci při práci s databází, tak jsem instanci třídy PDO, která pracuje s databází, zabalil do mnou vytvořené třídy DBConnectionManager, která je podle návrhového vzoru Singleton, jak jsem specifikoval v návrhu. Úkolem této třídy je zajistit, aby existovala pouze jediná instance třídy PDO s již nakonfigurovaným přístupem k databázi. Ostatní třídy získávají instanci PDO jedině přes třídu DBConnectionManager.

Databázové dotazy dělám zásadně přes metodu prepare z PDO. Tímto způsobem můžu při specifikaci dotazu vynechat parametry (nahradit je otazníky) a předat do dotazu až později.

Tohoto přístupu využívám hlavně pro vyvarování se bezpečnostních chyb typu SQLInjection, čímž bych mohl poškodit celou aplikaci a možná i nejen to.

Další problém, který jsem řešil při používání databáze, je zpracování chyb. Metoda execute, kterou se spouští nachystaný SQL dotaz už s předanými parametry, vrací logickou hodnotu false při neúspěchu. Takže testuju, jestli mi metoda execute nevrátí logickou hodnotu false, pokud ano, tak z objektu dotazu vytáhnu informace o chybě a metoda, ve které komunikuji s databází, vyhodí výjimku SQLException s patřičnou zprávou o selhání. Tímto způsobem jsem zajistil nešíření chyby napříč programem, neboť neodchycení SQLException zastaví běh s výpisem chyby, který jsem použil pro ladění.

5.5 Persistentní a dočasné objekty

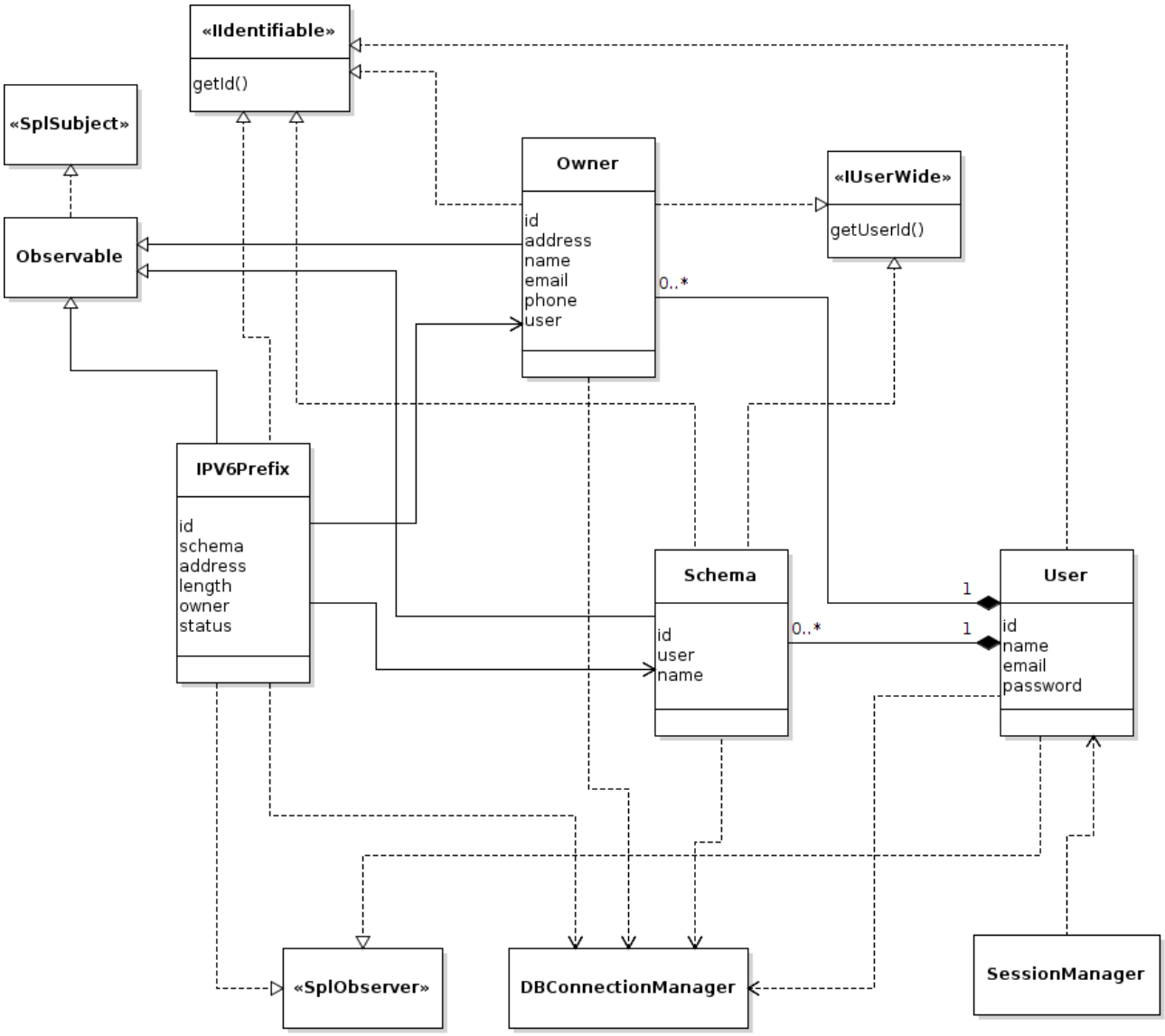
XML import pracuje s dočasnými objekty. Tyto objekty reprezentuji stejnými třídami jako ty, které v databázi jsou. Tohoto efektu jsem dosáhl tím, že objekt je ve výchozím stavu dočasný a zavoláním metody se z něj stane persistentní, tedy se uloží do databáze. Jelikož objekt nemůže být klíčem v poli, je nutné, aby i dočasné objekty měly identifikátory. Dočasným objektům jsem přiřadil dočasný identifikátor, který je odvozený od počitadla instancí. Avšak při uložení tohoto objektu do databáze se tento identifikátor změní (neboť není zajištěna dlouhodobá unikátnost dočasných identifikátorů), takže všude, kde se odkazovalo na objekt pomocí identifikátoru, se musí identifikátor aktualizovat, čehož je dosaženo pomocí dříve zmíněného návrhového vzoru Observer. Obdobně je řešena aktualizace při mazání.

Dalším problémem jsou konstruktory: je potřeba vědět, kdy vytváříme nový objekt a kdy pouze načítáme objekt z databáze. Jelikož programovací jazyk PHP nezná přetěžování metod, zvolil jsem přístup, kdy všechny takové třídy mají privátní konstruktor (nebo chráněný, pokud rodičovská třída má chráněný, protože v PHP na rozdíl třeba od Javy nelze mít privátní konstruktor ve třídě, která dědí od jiné třídy s veřejným nebo chráněným konstruktorem, platí zde bohužel stejné omezení jako pro normální metody) a operace jsou ze vnějšku třídy prováděny pouze skrze statické tovární metody s odpovídajícím názvem.

5.6 Práce s bloky adres

Blok adres reprezentuje třída IPV6Prefix. Pro uložení adresy používám celočíselný datový typ, který je v jazyce PHP na 64bitové platformě 64bitový. Ukládám pouze prvních 64 bitů adresy (viz body 6, 14 a 15 v návrhu). Výhodou tohoto řešení je možnost používat bitové operace jako bitové posuny, případně and, or nebo xor po bitech. Mezi nevýhody tohoto řešení je závislost na 64bitopvé instalaci běhového prostředí jazyka PHP (na 32bitových instalacích je celočíselný datový typ pouze 32bitový), případně nemožnost pracovat i s druhou půlkou adresy (celá adresa má totiž 128 bitů). Znaménkovost datového typu v jazyce PHP neřeším, protože používám výhradně bitové operace a ty také znaménko neřeší. V databázi je mírně odlišná situace, jelikož používám dotazy na rozsahy, ale problémy jsem zde obešel vhodným použitím funkce MOD, pomocí které problém převedu vždycky do rozsahu 0 – (2^{64}-1). O vzájemný převod textové podoby zápisu a interní reprezentace se starají patřičné statické metody třídy IPV6Prefix.

Další operací je vytvoření stromu, případně podstromu bloků adres. Tato operace je nutná pro sestavení jakýchkoliv vizualizací. Strom vytvářím pomocí dotazu na následovníky (tzn. všechny menší bloky adres, které spadají pod daný blok) a následným výběrem potomků. Tento proces rekurzivně dělám nad potomky. Výhodou tohoto řešení je, že se nemusím starat o integritu, ale mohly by se vyskytnout výkonnostní problémy, ale ty se během testů na ověřování funkčnosti neprojevily. Každopádně toto by bylo předmětem další optimalizace.



Obrázek 9: UML diagram popisující část Model z architektury MVC (zjednodušeno), je zde vidět využití návrhovévho vzoru Observer.

5.7 Filtry

Jelikož je třeba zjistit vzájemný výskyt vlastníků bloků ve schématech, vytvořil jsem pro tyto účely třídu OwnerSchemaOccurencyFilter. Tato třída následuje návrhový vzor Singleton a její metody složí pouze ke zjištění všech vlastníků vyskytujících se v daném schématu a všech schémat, kde daný vlastník figuruje.

5.8 XML import

Další důležitou součástí aplikace je XML import. Pro import XML používám knihovnu SimpleXML, která je přímo součástí jazyka PHP. Vedle toho má knihovna velmi jednoduché API, které se hodí právě pro moje účely. Import provádím ve třídě XmlParser, která pomocí knihovny SimpleXML převede vstupní data na objekty tříd mé aplikace. Třída ImportData pro změnu slouží pouze k seskupení takto získaných informací a je tedy podle návrhového vzoru Crate.

5.9 Zpracování sezení

Třída SessionManager zabezpečuje veškerou práci se sezením. Její hlavní účel je zjištění, který uživatel je aktuálně přihlášen a udržet dočasné objekty, které budou potřeba na některé z dalších webových stránek, v paměti.

5.10 Zpracování uživatelských akcí

O veškerý vstup od uživatele se stará část aplikace zvaná Controller, tzv. Master Controller. Tuto roli zastává soubor „index.php“, který podle parametru „page“ předá řízení Controlleru umístěném v adresáři „controller“. Ten následně vykoná úkony jako ověření, jestli uživatel je přihlášen, ověří uživatelský vstup a jestli má uživatel na danou akci oprávnění. Platnost uživatelem zadaných hodnot řeší tento Controller, ale integritní záležitosti (takový záznam už existuje) je podchycen integritními omezeními databáze (jako jsou unikátní klíče), které se Controller dozví vyhozením výjimky. Následně zavoláním patřičných metod z Modelu provede požadovanou operaci a pak pomocí View vytvoří výstup. Každé uživatelské operaci (zobrazení detailu vlastníka, zobrazení seznamu alokací, založení schématu, smazání alokace, atp.) náleží jeden tento Controller, jenž se stará o celou tuto operaci.

5.11 Informování uživatele o úspěchu a chybě

Všechny zprávy pro uživatele typu „úspěšně založen vlastník“, „schéma úspěšně smazáno“ nebo „tohoto vlastníka nelze zobrazit“ se zobrazují v rámečku pod navigací. Tento rámeček má barvu podle typu zprávy (chyby červeně, úspěšné operace zeleně, varování žlutě).

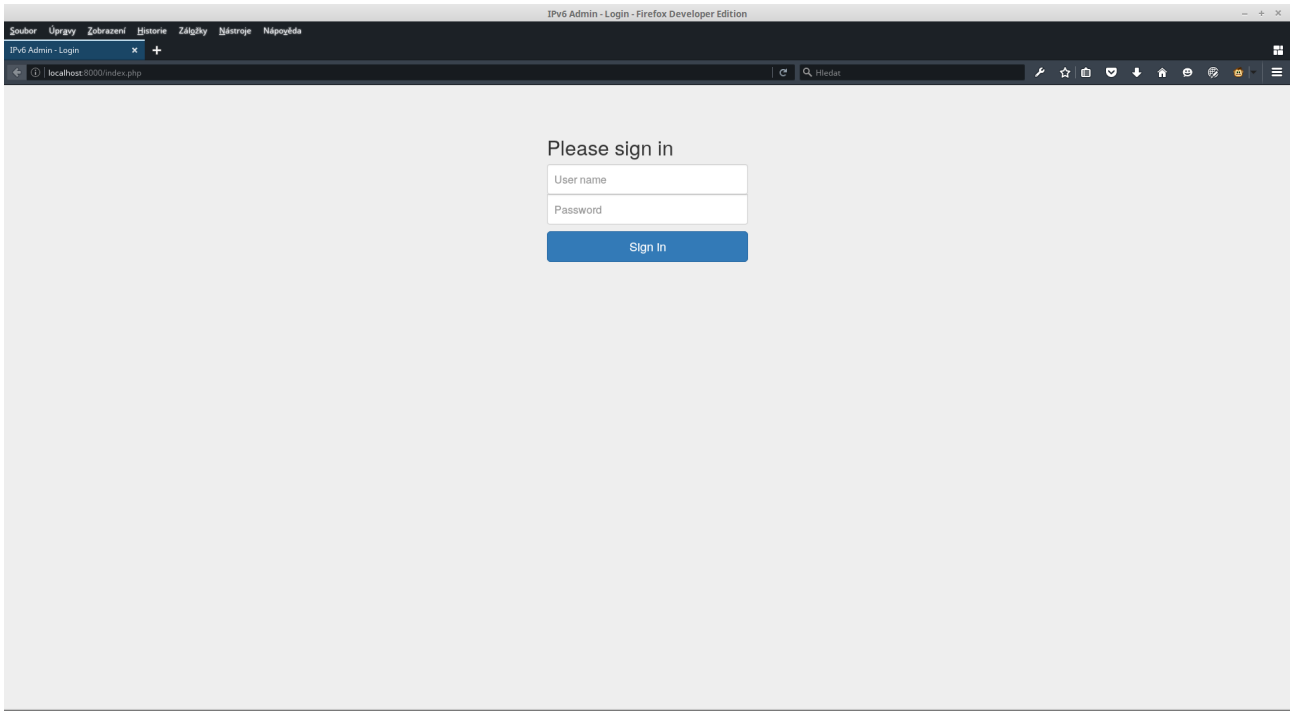
Celou problematiku řeším tak, že všechny vygenerované zprávy posbírá třída Message-Container, kterou jsem implementoval pomocí návrhového vzoru Singleton. Tímto postupem jsem zajistil, že posbírá opravdu všechny vzniklé zprávy v rámci jednoho požadavku. O jejich správné umístění na webové stránce se starají komponenty zodpovědné za rozvržení stránky.

5.12 Rozvržení stránky

Pro aplikaci jsem vytvořil dva typy rozvržení stránky, jeden pro přihlašovací stránku, druhý pro všechno ostatní. Přihlašovací stránka obsahuje pouze formulář pro přihlášení, který obsahuje pouze políčko pro uživatelské jméno, heslo a tlačítko pro přihlášení. Po přihlášení uživatel pokračuje na uvítací stránku.

Druhý způsob rozvržení stránky obsahuje horní nabídku a obsah pod ní. Horní nabídka obsahuje název webu, dvě rozbalovací nabídky, jednu pro správu schémat, druhou pro správu vlastníků a nakonec informaci o tom, kdo je přihlášen s možností odhlásit se.

Pod nabídkou se objevují zprávy o úspěchu nebo chybě a pod nimi samotný obsah stránky. Pro implementaci této struktury jsem vytvořil třídu PageBuilder podle návrhového vzoru Singleton, neboť na jeden požadavek můžeme zobrazit pouze jednu webovou stránku. PageBuilder se stará o zobrazení všech společných částí (hlavička stránky, nabídka, vložení skriptů, atd.) a do toho vloží obsah, který je generován jednotlivými třídami. Některé typy obsahu dokonce mohou obsahovat jiný obsah, jedná se o kontejnery.



Obrázek 10: Snímek z obrazkovky ukazující přihlašovací obrazovku

5.13 Typy obsahu

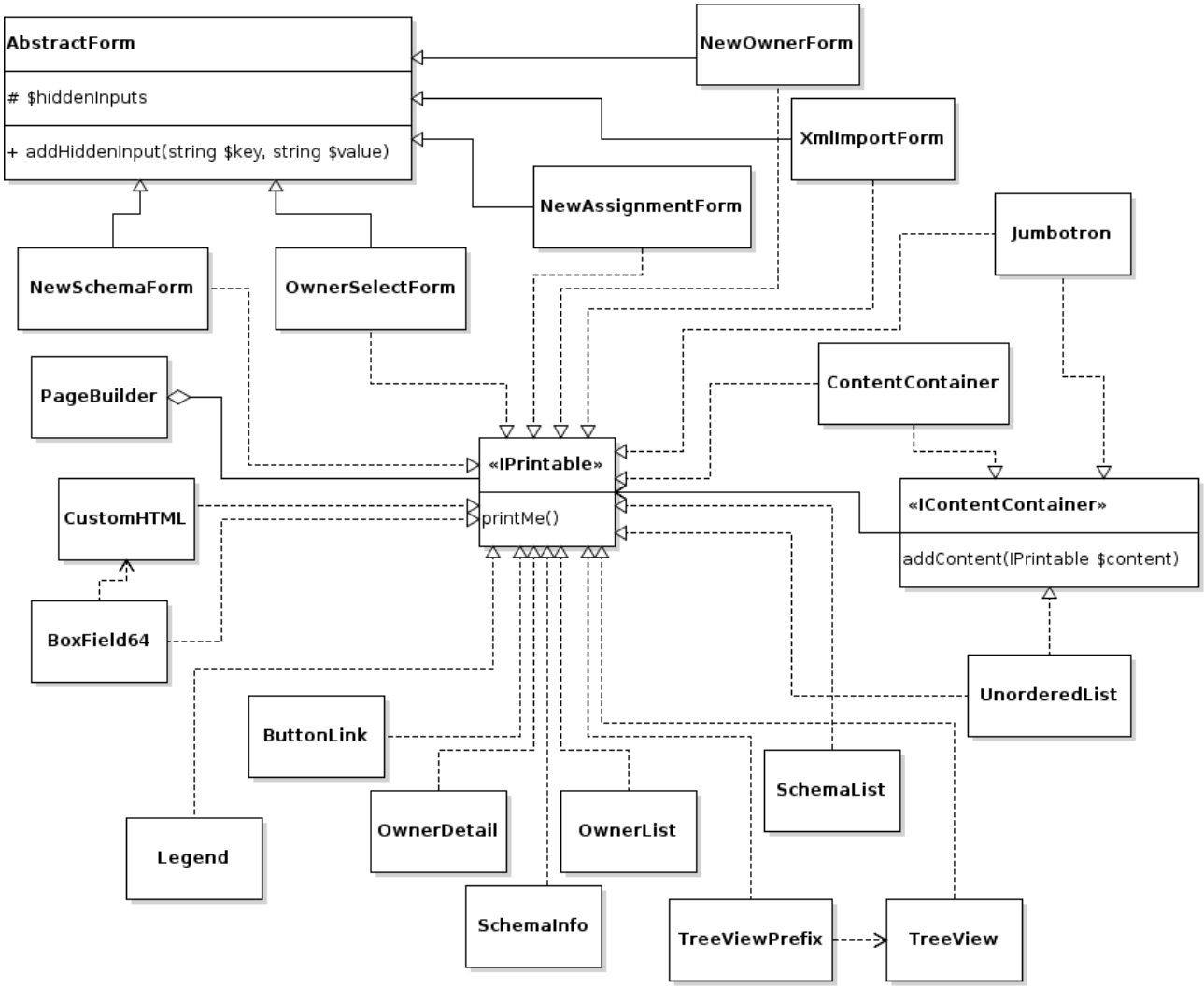
V aplikaci používám několik typů obsahu, z toho některé mají určité specifické vlastnosti. Dříve jsem zmínil kontejnery, které do sebe umí vložit další obsah. Tato vlastnost je užitečná ve chvíli, když potřebuji místo jednoho obsahu umístit za sebe obsahů víc, jako jsou například detaily zobrazovaného objektu a pod ním seznam výskytu. Druhým takovým typem jsou formuláře, kde se mi osvědčilo mít možnost do formuláře přidat skryté pole s fixní hodnotou. Tohoto mechanismu využívám pro přenos některých informací přes vícekrokové operace, kde uživatelem zadané hodnoty v předchozím kroku se takto promítnou do formuláře. Výhodou je, že aplikace si nemusí pamatovat tyto hodnoty jiným způsobem na straně serveru, a tudíž tuto práci realizuje klient.

Účel jednotlivých obsahových tříd bude popsán dále u popisu jednotlivých souborů. Tyto třídy jsem pojmenoval buď podle jejich primární funkce, podle jejich vzhledu, nebo podle komponenty z frameworku Bootstrap, kterou vykreslují. Editační formuláře jsou řešeny

pomocí stejných tříd, jako vytvářecí, jenom jsem přidal argument pro editační režim. Nyní detailně popíšu části, které nejsou implementačně triviální.

5.14 Našeptávač

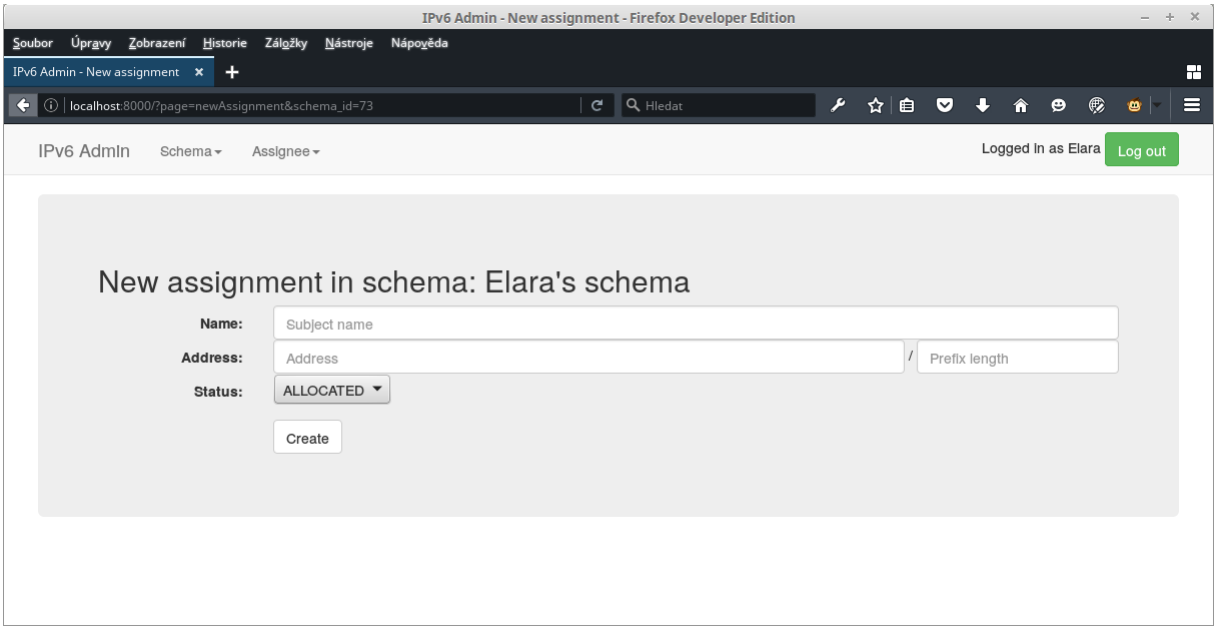
Ve své aplikaci jsem použil našeptávač. Našeptávač se skládá ze dvou částí: frontend a backend. Frontend se zabývá přenosem právě psaného textu do backendu a zobrazením výsledků backendu, kdežto úlohou backendu je z obdrženého právě psaného textu udělat seznam kandidátů, který zase zobrazí frontend.



Obrázek 11: UML diagram popisující hierarchii jednotlivých prvků pro prezentaci dat

K realizaci frontendu jsem použil nakonec javascriptovou knihovnu jQuery-ui. Než jsem vůbec sáhnul po této knihovně, tak jsem experimentoval s knihovnou Bootstrap3-typeahead, z důvodu lepší vzhledové provázanosti s použitým frameworkem Bootstrap. Bohužel, nepodařilo se mi to provázat s další knihovou Bloodhound, která sloužila pro komunikaci se vzdáleným seznamem našeptávaných hodnot (backendem), kdežto zvolené řešení v podobě jQuery-ui s tímto nemělo žádné problémy. Z backendu posílám informace ve formátu JSON, právě pro bezvadnou podporu tohoto formátu v JavaScriptu.

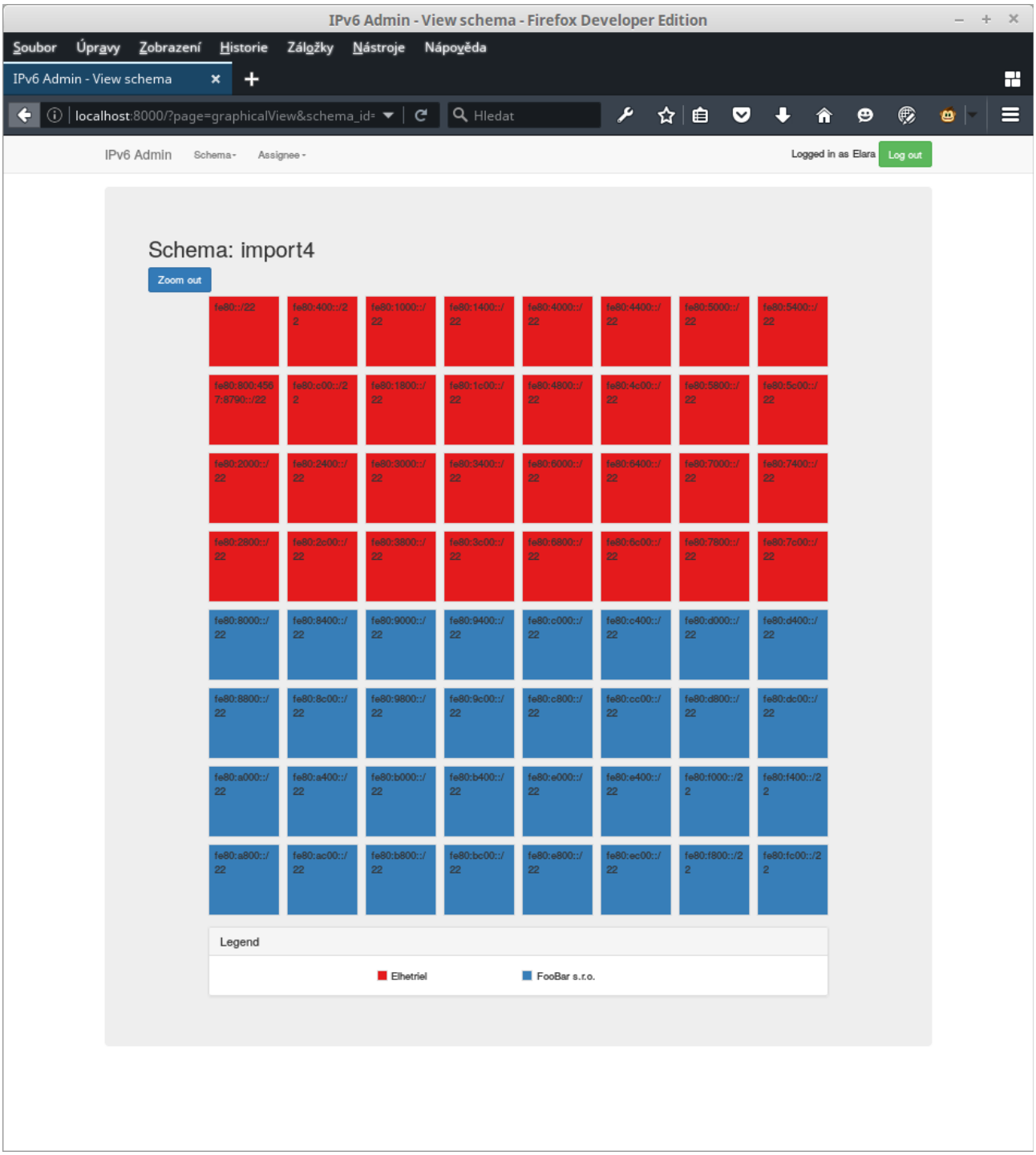
Backend našeptávače jsem tvořil s ohledem na řešení částečné shody. Důvodem tohoto přístupu je, že když uživatel zadá „Firma s. r. o.“, tak aby mu našeptávač nabídl i „Firma a. s.“ a obdobně opravil i drobné překlepy. Samostatné hledání přibližné shody jsem řešil externím programem agrep, který je na tento problém optimalizován. Nevýhoda tohoto řešení je závislost na platformě, na které externí program běží, a nutnost komunikace s tímto programem. Další nevýhodou je nutnost načíst celý uživatelův seznam vlastníků do aplikace a předat ho tomuto programu ve vhodné formě a zpátky přečíst jeho výstup, který se převede do formátu JSON, aby šel poslat zpátky do frontendu k zobrazení. Jelikož agrep je program stvořený pro příkazovou řádku, který pracuje se standardním vstupem a standardním výstupem, kde na jednom řádku je jeden záznam. Vyhledávací parametry se předávají jako argumenty programu. Problém standardního vstupu jsem řešil pomocí bashovského operátoru „<<<“, který pošle do standardního vstupu programu mnou zvolený řetězec. Tento řetězec jsem sestavil tak, že jsem napřed vložil identifikátor záznamu, pak oddělovač, potom název záznamu, následovaný znakem pro konec řádku a dalším záznamem. Důvodem použití identifikátoru bylo snazší zpětné spárování s objekty, mezi kterými vyhledávám, jelikož mi pak agrep vždycky vrátil identifikátor s názvem. Pomocí identifikátorů jsem pak mohl tento systém použít i na jiných místech (například zpracování vstupu z formulářového pole, kde byl použit našeptávač, nebo při řešení konfliktů vlastníků u XML importu). Z výsledků jsem sestavil JSON, který jsem poslal do frontendu.



Obrázek 12: Vzhled formuláře pro zadávání nových alokací, pro vyplnění názvu vlastníka pomáhá našeptávač založený na principu částečné shody. Konkrétního vlastníka uživatel vybere v následujícím kroku ze seznamu, kde vidí i kontaktní údaje.

5.15 Grafická vizualizace

Nejdůležitějším prvkem aplikace je grafická vizualizace. Ostatně to je hlavní důvod, proč celá aplikace vznikla. O grafický vzhled se stará třída BoxField64, která realizuje vizualizaci pomocí čtvercové sítě o 64 čtvercích a třída Legend, která má na starost legendu popisující použitou symboliku. Uspořádání jednotlivých čtverců symbolizující jednotlivé menší bloky adres je zachováno dle návrhu. Každý z těchto čtverců je buď vybarven barvou svého majitele (v případě více majitelů v hierarchii je vzat ten z hierarchie, který pokrývá celý blok adres a zároveň je v hierarchii co nejníže), nebo je šrafovaný v případě, že veškeré alokace v daném bloku jsou hierarchicky níže, než je adresa daného bloku. V případě, že v bloku žádné alokace



Obrázek 13: Snímek obrazovky ukazující grafickou vizualizaci.

nejsou, je blok nechán prázdný (bílý). Při najetí myši na čtverec, se čtverec zvýrazní a při kliknutí se objeví okno s podrobnějšími informacemi. Toto okno jsem vytvořil za pomocí Bootstrapové komponenty „Modal“. Toto okno lze zavřít buď k tomu určeným tlačítkem v okně nebo kliknutím myši kamkoliv do stránky mimo toto okno. V okně je rovněž umístěno tlačítko pro přiblížení se v hierarchickém pohledu. Pro oddálení se je určeno tlačítko nad celou čtvercovou sítí. Tato tlačítka nejsou k dispozici v krajních polohách hierarchického pohledu, aby se nešlo dostat mimo tyto meze.

Jako výchozí úroveň hierarchického pohledu jsem implementoval takovou úroveň, aby z ní byly vidět všechny alokace a zároveň byla všem alokacím co nejblíže a zároveň uvnitř mezí.

Barvy jednotlivých vlastníků jsou dynamicky brány z množiny, kterou jsem implementoval za pomocí třídy ColorManager. Tímto jsem zajistil unikátní barvu vlastníků, pokud jich není v jednom hierarchickém pohledu více, než je barev v poolu.

5.16 Popis jednotlivých adresářů a souborů

V této části se zaměřím na popis jednotlivých souborů a adresářů a zmínění účelu jednotlivých částí aplikace. Soubory, jejichž jméno začíná velkým písmenem a končí koncovkou php, obsahují vždy pouze stejnojmennou třídu. Kořenový adresář aplikace obsahuje následující adresáře a soubory:

- config – Obsahuje konfiguraci jako přístupové údaje k databázi.
- controller – Obsahuje všechno, co se týče komponenty Controller z architektury MVC až na soubor „index.php“ (z technických důvodů).
- model – Obsahuje Model komponentu MVC, zde je business logika
- resources – Obsahuje všechny statické objekty, jako styly, JavaScript a soubory frontendových frameworků a frontendových knihoven, tj. všechny soubory, které se přenášejí ke klientovi beze změny.
- test – Obsahuje unit testy (podrobněji v kapitole o testování).
- view – Obsahuje všechny soubory z komponenty View z architektury MVC, jedná se o soubory zabývající se prezentací dat.
- index.php – Tzv. „Master Controller“, vstupní bod do aplikace.

V adresáři contoller jsou definovány jednotlivé akce, které uživatel může udělat, každou jsem realizoval jedním souborem, také tento seznam popisuje, co aplikace přesně umí:

- assignmentDelete.php – rušení alokace
- assignmentDetail.php – zobrazení detailů alokace
- assignmentEdit.php – změna alokace
- default.php – výchozí akce (přihlášení nebo uvítací obrazovka)

- deleteOwner.php – smazání vlastníka, automaticky maže i všechny jeho alokace
- deleteSchema.php – smazání schématu, automaticky maže i všechny alokace ve schématu
- graphicalView.php – grafický pohled na schéma, detailně popsán dříve
- listAllocations.php – seznam alokací vlastníka ve schématu
- login.php – přihlášení se
- logout.php – odhlášení se
- newAssignment.php – nová alokace
- newOwner.php – založení vlastníka
- newSchema.php – založení schématu
- ownerDetail.php – informace o vlastníkovi
- ownerEdit.php – editace vlastníka
- ownerList.php – seznam vlastníků vytvořených uživatelem
- ownerListJSON.php – slouží jako backend pro našeptávač
- schemaEdit.php – úprava schématu – změna názvu
- schemaOwners.php – seznam vlastníků vyskytujících se ve schématu
- treeView.php – zobrazení alokací ve schématu jako strom
- viewSchema.php – zobrazení informací o schématu, výchozí bod pro operace se schématem
- xmlExport.php – export do XML
- xmlImport.php – import z XML

Adresář model zabývající se business logikou obsahuje tyto soubory a adresáře, uvádím hlavně z toho důvodu, že ne všechno se vešlo do patřičného UML diagramu:

- db – Obsahuje soubor DBConnectionManager.php se stejnojmennou třídou zajišťující přístup k databázi.
- lib – Obsahuje ty části, které by se daly použít i pro řešení jiných problémů, obsahuje následující soubory a adresáře:
    - exception – Obsahuje mnou vytvořené použité výjimky:
        - AnonymousUserException.php – Výjimka vyhazována při přístupu nepřihlášeného uživatele do sekce, kde je vyžadováno přihlášení.

        - InvalidCredentialsException.php – Výjimka vyhazována v případě neplatných přihlašovacích údajů.
        - NoSuchElementExcpetion.php – Výjimka obvykle vyhazována v případech, kdy by metoda ráda vrátila null, ale nemůže kvůli typové kontrole.
        - NotImplementedException.php – Výjimka vyhazována v případě, že metoda není pro daný případ implementována.
        - PartiallyOwnedException.php – Výjimka vyhazována v případě, že daný blok adres má být vyobrazen šrafováním.
    - Delete.php – Slouží pro zaslání zprávy v rámci návrhového vzoru Observer, že daný objekt byl smazán.
    - IdChangeCrate.php – Slouží pro zaslání zprávy v rámci návrhového vzoru Observer, že daný objekt změnil identifikátor.
    - IIdentifiable.php – Obsahuje rozhraní pro polymorfismus, které značí, že objekt má identifikátor.
    - IUserWide.php – Obsahuje rozhraní pro polymorfismus, které značí, že objekt je vázaný na konkrétního uživatele.
    - Observable.php – Abstraktní třída pro návrhový vzor Observer.
- AllocationStatus.php – Výčet různých stavů alokace (alokováno, rezervováno).
- ImportData.php – Třída podle návrhového vzoru Crate pro shluknutí importovaných dat z XML
- IPV6Prefix.php – Třída obsahující logiku okolo bloků adres.
- Owner.php – Třída reprezentující vlastníka bloků adres.
- OwnerSchemaOccurencyFilter.php – Filtr na zjištění vlastníků přítomných ve schématu a naopak schémat, kde se daný vlastník nachází, zamezuje cyklickým vazbám.
- SessionManager.php – správce sezení a superglobálního pole $_SESSION.
- Schema.php – Třída reprezentující schémata.
- TreeNode.php – Třída pro vytváření stromů bloků adres.
- User.php – Třída reprezentující uživatele.
- XmlParser.php – Třída pro převod importovaného XML na objekty mnou vytvořených tříd.

Jako poslední adresář, kterému budu věnovat detailnější popis, je adresář view, který se zabývá prezentací dat:

- commonPages – Adresář obsahující už připravené stránky, kam se běžně uživatel přesměrovává, jako přihlašovací a uvítací obrazovku.
- content – Adresář obsahující jednotlivé bloky obsahu:
  - BoxField64.php – Vysokoúrovňová komponenta obsahující základ pro grafickou vizualizaci alokací.
  - ButtonLink.php – Nízkoúrovňová komponenta sloužící pro zobrazení odkazu jako tlačítko.
  - ContentContainer.php – Slouží pro pouhé zobrazení více obsahů místo jednoho.
  - CustomHTML.php – Slouží pro zobrazení vlastního HTML, je použita akorát pro zobrazení čistého textu (použití pro cokoliv složitějšího by bylo v rozporu s architekturou MVC. Je použita i pro prázdný obsah, kde je z implementačních důvodů potřeba.
  - Jumbotron.php – Nízkoúrovňová komponenta reprezentující prvek Jumbotron z použitého frontend frameworku Bootstrap.
  - Legend.php – Slouží k zobrazení legendy v grafické vizualizaci.
  - NewAssignemntForm.php – Formulář pro zakládání a změnu alokací.
  - NewOwnerForm.php – Formulář pro zakládání a změnu vlastníků bloků adres.
  - NewSchemaForm.php – Formulář pro zakládání schémat.
  - OwnerDetail.php – Informace o vlastníkovi.
  - OwnerList.php – Seznam vlastníků.
  - OwnerSelectForm.php – Formulář pro výběr patřičného vlastníka, obvykle použito pro výběr z možností vrácených programem agrep.
  - SchemaInfo.php – Informace o schématu.
  - SchemaList.php – Seznam schémat.
  - TreeView.php – Stromová vizualizace schématu.
  - TreeViewPrefix.php – Stromová vizualizace pouze části schématu, týkající se určitého bloku adres.
  - UnorderedList.php – Nízkoúrovňová komponenta pro nečíslovaný seznam.
  - XmlImportForm.php – Formulář pro nahrání XML souboru pro import.
- AbstractForm.php – Abstraktní třída pro formuláře umožňující přidat do formulářů skrytá pole s předdefinovanými hodnotami.

- ButtonStyle.php – Výčet typů tlačítek podporovaných použitým frontend frameworkem Bootstrap.
- ColorManager.php – Množina barev pro grafickou vizualizaci.
- IContentContainer.php – Rozhraní umožňující polymorfismus u druhů obsahu, které do sebe umí vkládat jiný obsah.
- IPrintable.php – Rozhraní, které implementuje každý typ obsahu.
- Message.php – Reprezentuje zprávu o úspěchu nebo chybě uživatelovy operace.
- MessageContainer.php – Drží všechny zprávy o úspěchu nebo chybě, které se mají vypsat.
- MessageType.php – Výčet typů různých zpráv o úspěchu nebo chybě. Důležité pro vizuální odlišení.
- PageBuilder.php – Popisuje základní rozvržení skoro všech zobrazovaných stránek. Stará se o všechny jejich společné věci.
- XmlExport.php – Zajišťuje export do formátu XML.

6 Testování webové aplikace

Pro testování aplikace jsem použil dvě metody, každou z nich pro jinou část aplikace. Pro testování business logiky jsem použil automatické testy za pomocí frameworku PHPUnit, ostatní části aplikace jsem testoval uživatelsky.

Automatické testování pomocí frameworku PHPUnit se mi ukázalo jako velmi důležité, neboť mě seznámily s mnoha nepříjemnými vlastnostmi jazyka PHP verze 7.0, které jsem popsal v rámci implementace. Navíc toto mi pomohlo otestovat business logiku aplikace samostatně, tedy dříve, než jsem vyvinul cokoliv týkající se vizuální stránky aplikace. Velmi se mi osvědčilo testování okolo práce s programem agrep a tím jsem dosáhl správné komunikace s tím programem (jelikož jsou zde mnohá technologická úskalí) a relevanci jeho výsledků. Rovněž mi toto pomohlo pro zúžení prostoru pro hledání příčiny chyby v rámci uživatelského testování.

Uživatelské testování jsem prováděl následně na částech systému, které se obtížněji automaticky testují. Testoval jsem, zda daná webová stránka vypadá tak, jak má, zda jsou zobrazeny správné údaje, zda odkazy směřují tam, kam mají a nejsou mrtvé, rovněž jsem testoval průchod různých uživatelských operací při různých scénářích. Tyto testy jsem prováděl pomocí webového prohlížeče Firefox Developer Edition, o kterém jsem se zmiňoval při návrhu aplikace. Tento prohlížeč se mi zejména osvědčil při ladění našeptávače a grafické vizualizace.

7 Závěr

Seznámil jsem se s problematikou přidělování adres IPv6 a navrhl jsem, implementoval a otestoval webovou aplikaci pro jejich správu. Rád bych zde zmínil, do jaké míry se mi povedlo splnit jednotlivé požadavky na aplikaci:

1. Musí se jednat o webovou aplikaci, kterou bude uživatel používat pomocí webového prohlížeče.
   - Splněno, navrhl jsem a implementoval webovou aplikaci, se kterou uživatel pracuje prostřednictvím webového prohlížeče.
2. Každý uživatel musí mít svá data, nesdílená s ostatními uživateli.
   - Splněno, aplikace vyžaduje, aby se uživatel pro jakoukoliv práci s ní přihlásil, veškerý obsah vytvořený uživatelem ostatní uživatelé nevidí.
3. Musí být možno přidávat, rušit a měnit alokace.
   - Splněno.
4. Musí být možno přidávat, rušit a měnit informace o vlastnících bloků adres.
   - Splněno.
5. Aplikace musí být schopná graficky zobrazit aktuální stav (tak, jak ho uživatel zadal), a to v libovolné úrovni hierarchického pohledu.
   - Splněno, grafická vizualizace se odehrává na 64 čtvercích, uspořádanými do sítě 8×8 čtverců. Pro změnu úrovně hierarchického pohledu jsem udělal tlačítka.
6. Aplikace musí být schopna pracovat s bloky adres velikosti aspoň /64.
   - Splněno, aplikace ukládá prvních 64 bitů adresy, což postačuje pro bloky velikosti /64.
7. Aplikace musí umět data exportovat do formátu XML a také z téhož formátu importovat.
   - Splněno, pro export i import použit formát vytvořený v rámci magisterského projektu.
8. Aplikace by měla mít možnost spárovat informace o vlastnících z importu s těmi již v databázi.
   - Splněno, v případě přesné shody toto udělá aplikace automaticky, v případě neúplné vyzve uživatele.
9. Aplikace by měla uživateli umožnit mít více kontextů, ve kterých přiděluje, mění a ruší alokace bloků adres a tyto kontexty i pojmenovat, přejmenovat nebo mazat.
   - Splněno, tento problém implementován pomocí tzv. schémat.

10. Aplikace by měla být schopna ukládat stavy (alokováno, rezervováno) alokací bloků adres.
    - Splněno, pro jednotlivé možné stavy jsem implementoval výčet.
11. Aplikace může umožnit uživateli specifikovat v rámci grafického zobrazení alokací nastavit vlastníkovi barvu, kterou bude v grafických zobrazeních vždy vyobrazen.
    - Nesplněno, aplikace barvy vlastníkům přiděluje dynamicky, při jednotlivém zobrazení z poolu. Navíc se jednalo o nápad vedoucího práce až v době blížícího se odevzdání práce.
12. Aplikace může umožnit uživateli sloučit dva kontexty přidělování bloků adres do jednoho.
    - Nesplněno, na realizaci tohoto bodu nezbyl bohužel čas.
13. Aplikace by mohla zobrazovat obsazenost daného bloku adres například metrikou HD-Ratio podle RIPE NCC.
    - Nesplněno, RIPE NCC toto měří vůči objektům velikosti /56, v nižší hierarchické úrovni tudíž tato metrika zcela postrádá smysl. Navíc se jednalo o nápad vedoucího v době blížícího se odevzdávání práce.
14. Aplikace nemusí být schopna pracovat s bloky menšími než /64
    - Této vlastnosti jsem využil pro možnost reprezentace adresy pomocí 64 bitového celého čísla (a druhou půlku adresy jsem zahodil). Tento postup usnadnil implementaci některých operací.
15. Postačuje, když server bude fungovat na jednom běžném operačním systému a běžné počítačové architektuře (například Linux nad amd64)
    - Této vlastnosti jsem využil, protože celočíselný datový typ v jazyce PHP o délce 64 bitů je dostupný pouze na 64bitových platformách. Dále využívám externí program agrep pro nalezení částečné shody, který používám v rámci shellu bash pro předání vstupu. Tento postup je možný pod operačním systémem Linux, který je běžně používán v serverovém nasazení.

Úspěšně jsem splnil všechny body, u kterých je uvedeno „musí“, čímž jsem splnil zadání diplomové práce. Navíc jsem splnil některé bonusové cíle.

Jsem si plně vědom, že na práci je pořád prostor pro zlepšování. Například by stálo za to provést výkonnostní testy a podle nich navrhnout a implementovat optimalizace. Rovněž by stálo za to implementovat zbylé nesplněné bonusové cíle.

Stálo by za to vytvořit algoritmus pro nalezení vhodného volného bloku adres dané velikosti v určitém rozsahu. Tím usnadnit život administrátorům, aby takový blok adres nemuseli sami hledat.

Reference

[1] SATRAPA, Pavel. *IPv6: internetový protokol verze 6.* 3., aktualiz. a dopl. vyd. Praha: CZ.NIC, c2011, 407 s. CZ.NIC. ISBN 978-80-904248-4-5. Dostupné z: http://knihy.nic.cz/files/nic/edice/pavel_satrapa_ipv6_2012.pdf

[2] KAUSHIK, Das. *IPv6 -The History and Timeline* [online]. [cit. 2015-05-22]. Dostupné z: http://www.ipv6.com/articles/general/timeline-of-ipv6.htm

[3] SATRAPA, Pavel. *Historie a Současnost IPv6* [online]. [cit. 2015-05-22]. Dostupné z: http://www.europen.cz/Proceedings/38/prezentace.pdf

[4] Topologie IPv6 sítě CESNET2. *CESNET* [online]. [cit. 2015-05-25]. Dostupné z: http://www.cesnet.cz/sluzby/pripojeni/ipv6/topologie/

[5] *What Does ICANN Do?* [online]. [cit. 2015-05-22]. Dostupné z: https://www.icann.org/resources/pages/what-2012-02-25-en

[6] Number Resources. *IANA* [online]. [cit. 2015-05-22]. Dostupné z: https://www.iana.org/numbers

[7] *What is an RIR?* [online]. [cit. 2015-05-22]. Dostupné z: http://whatismyipaddress.com/rir

[8] What is a Local Internet Registry (LIR)? *RIPE NCC* [online]. [cit. 2015-05-22]. Dostupné z: https://www.ripe.net/manage-ips-and-asns/resource-management/faq/independent-resources/phase-three/what-is-a-local-internet-registry-lir

[9] LEDVINA, Jiří. *Protokol IP verze 6* [online]. [cit. 2015-05-22]. Dostupné z: http://www.kiv.zcu.cz/~ledvina/Prednasky-PDS-2007/11a_ipv6-psi.pdf

[10] PETERKA, Jiří. *Subnetting, supernetting a CIDR* [online]. 1999 [cit. 2015-05-22]. Dostupné z: http://www.earchiv.cz/anovinky/ai1681.php3

[11] Proč není NAT totéž co firewall. *Root.cz: Informace nejen ze světa Linuxu* [online]. 2007 [cit. 2015-05-22]. Dostupné z: http://www.root.cz/clanky/proc-neni-nat-totez-co-firewall/

[12] *Implementace protokolu IPv6 v síti CESNET2* [online]. 2003 [cit. 2015-05-22]. Dostupné z: http://akce.fs.vsb.cz/2003/asr2003/Proceedings/papers/313.pdf

[13] What is DHCP? *Indiana University: Knowledge Base* [online]. 2015 [cit. 2015-05-22]. Dostupné z: https://kb.iu.edu/d/adov

[14] IPv6 Addressing scheme. *CESNET* [online]. [cit. 2016-05-10]. Dostupné z: https://www.cesnet.cz/services/ip-connectivity-ip/ipv6/ipv6-addressing-scheme/?lang=en

[15] Assessment Criteria for Initial IPv6 Allocation. *RIPE NCC: RIPE NETWORK COORDINATION CENTRE* [online]. 2015 [cit. 2016-05-08]. Dostupné z: https://www.ripe.net/manage-ips-and-asns/ipv6/request-ipv6/assessment-criteria-for-initial-ipv6-allocation

[16] Request IPv6. *RIPE NCC: RIPE NETWORK COORDINATION CENTRE* [online]. 2010, 07 Oct 2015 [cit. 2016-05-10]. Dostupné z: https://www.ripe.net/manage-ips-and-asns/ipv6/request-ipv6

[17] IPv6 Address Allocation and Assignment Policy. *RIPE NCC: RIPE NETWORK COORDINATION CENTRE* [online]. 2015 [cit. 2016-05-10]. Dostupné z: https://www.ripe.net/publications/docs/ripe-655

[18] IPv6 Address Space Policy For Internet Exchange Points. *RIPE NCC: RIPE NETWORK COORDINATION CENTRE* [online]. 2009 [cit. 2016-05-10]. Dostupné z: https://www.ripe.net/publications/docs/ripe-451

[19] *Bootstrap* [online]. [cit. 2016-05-10]. Dostupné z: http://getbootstrap.com/

[20] *Foundation: The most advanced responsive front-end framework in the world.* [online]. ZURB [cit. 2016-05-10]. Dostupné z: http://foundation.zurb.com/

[21] *ECMAScript 6: New Features: Overview & Comparison* [online]. [cit. 2016-05-10]. Dostupné z: http://es6-features.org

[22] KANGAX, WEBBEDSPACE a ZLOIROCK. *ECMAScript compatibility table* [online]. [cit. 2016-05-10]. Dostupné z: https://kangax.github.io/compat-table/es6/

[23] *Babel is a JavaScript compiler: Use next generation JavaScript, today.* [online]. [cit. 2016-05-10]. Dostupné z: https://babeljs.io/

[24] *Firefox Developer Edition: Vytvořen pro ty, kteří utváří web: Jediný prohlížeč vytvořený pro vyvojáře, jako jste vy.* [online]. [cit. 2016-05-10]. Dostupné z: https://www.mozilla.org/cs/firefox/developer/

[25] BERNARD, Borek. Úvod do architektury MVC. In: *Zdroják.cz* [online]. 2009 [cit. 2016-05-10]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[26] Hardwarový a softwarový průzkum: April 2016. *STEAM* [online]. [cit. 2016-05-10]. Dostupné z: http://store.steampowered.com/hwsurvey/

[27] Migrating from PHP 5.6.x to PHP 7.0.x. *PHP* [online]. [cit. 2016-05-10]. Dostupné z: http://php.net/migration70

[28] HUJER, Martin. Jaké novinky přinese PHP 7. In: *Zdroják.cz* [online]. 2015 [cit. 2016-05-10]. Dostupné z: https://www.zdrojak.cz/clanky/jake-novinky-prinese-php-7/

[29] Turbocharging the web with PHP 7. *Zend* [online]. [cit. 2016-05-11]. Dostupné z: http://www.zend.com/en/resources/php7_infographic

[30] PALÍK, Václav. *NÁVRH VÝMĚNNÉHO DATOVÉHO FORMÁTU PRO APLIKACI PRO SPRÁVU ADRESNÍHO PROSTORU IPV6*. Liberec, 2015. Magisterský semestrální projekt. Technická univerzita v Liberci. Vedoucí práce Pavel Satrapa.

Příloha A – obsah přiloženého CD

- Elektronická verze této práce ve formátu PDF.
- Zdrojové kódy programu.

Příloha B – navržený výměnný datový formát

XML Schema pro výměnný datový formát používaný pro import a export. [30]

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:complexType name="ownerType">
		<xs:sequence>
			<xs:element name="name" type="xs:token" />
			<xs:element name="email" type="xs:token" />
			<xs:element name="phone" type="xs:token" />
			<xs:element name="address" type="xs:token" />
			<xs:element name="owner" type="ownerType" minOccurs="0"
				maxOccurs="unbounded" />
		</xs:sequence>
		<xs:attribute name="id" use="required" />
	</xs:complexType>
	<xs:complexType name="prefixType">
		<xs:attribute name="address" use="required" />
		<xs:attribute name="length" use="required" />
		<xs:attribute name="owner" use="required" />
		<xs:attribute name="state" use="required" />
	</xs:complexType>
	<xs:element name="ipv6data">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="prefixes">
					<xs:complexType>
						<xs:sequence>
							<xs:element name="prefix"
	minOccurs="1" maxOccurs="unbounded" type="prefixType" />
						</xs:sequence>
					</xs:complexType>
				</xs:element>
				<xs:element name="owners">
					<xs:complexType>
						<xs:sequence>
							<xs:element name="owner"
	minOccurs="1" maxOccurs="unbounded" type="ownerType" />
						</xs:sequence>
					</xs:complexType>
				</xs:element>

			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```