VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ADMINISTRAČNÍ SYSTÉM HLASOVÁNÍ ZASTUPITELSTEV

COUNCIL VOTING MANAGEMENT SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ŠTĚPÁN KREJČÍŘ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. KRISTÝNA ZAKLOVÁ**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

161358

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Krejčíř Štěpán, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Informační systémy a databáze
Název:	**Administrační systém hlasování zastupitelstev**
Kategorie:	Informační systémy
Akademický rok:	2024/25

Zadání:

1. Seznamte se s problematikou zastupitelstev a dat z jejich hlasování. Soustřeďte se na datové modely a ukládaná data.
2. Prostudujte problematiku vývoje informačních systémů. Dále se zaměřte na průzkum současných redakčních systémů s důrazem na administrační prostředí (správa databází, záloha dat apod.).
3. Seznamte se se systémem hlasování zastupitelstev. Analyzujte současný stav a požadavky na administrační modul.
4. Na základě zjištěných poznatků navrhněte rozšíření systému o administrační část, která bude umožňovat činnosti definované v bodu 3.
5. Po konzultaci s vedoucí navržené rozšíření systému implementujte.
6. Vytvořené řešení otestujte a zhodnoťte dosažené výsledky.

Literatura:

- Rainer, R. K., Prince, B., Cegielski, C. G. 2013. *Introduction to Information Systems* (5th. ed.). Wiley Publishing. ISBN 978-1-118-67436-9.
- Richardson, L., Amundsen, M. *RESTful Web APIs*. Sebastopol: O'Reilly, 2013. ISBN 978-1-4493-5806-8.
- Zaklová, K. (2023). *Analýza a vizualizace dat z hlasování Zastupitelstva města Brna*. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/146480

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Zaklová Kristýna, Ing.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	22.10.2024

Abstrakt

Cílem této práce bylo rozvinout stávající administrační systém projektu zabývajícího se vizualizací hlasovacích dat o možnost správy databází a o obecnější podporu různých typů samosprávných orgánů. Pomocí frameworku Flask a knihovny React bylo vytvořeno rozšíření dovolující přímo v administračním rozhraní vytvářet, mazat, exportovat a importovat relační databáze. Do systému byla dále integrována NoSQL databáze MongoDB, do níž jsou exporty relačních databází zálohovány. Obecnější podpory dalších typů orgánů bylo dosaženo úpravou stávajících komponent. Hlavním přínosem práce by měl být snadnější přenos databází mezi jednotlivými větvemi projektu a snížení rizika ztráty dat. Zobecněná správa by pak měla dovolit jednodušší integraci dalších typů orgánů do systému.

Abstract

The goal of this work was to extend the existing administration system of a voting data visualisation project by incorporating database management capabilities and general support for different types of local authorities within the system. Using the Flask framework and the React library, an extension was created to allow the creation, deletion, export and import of relational databases directly in the administration interface. The NoSQL database MongoDB was also integrated into the system, to which relational database exports are backed up. General support for other types of authorities was achieved by modifying existing components. The main benefit of this work should be easier transfer of databases between different branches of the project and reduced risk of data loss. The generalised management should allow easier integration of other types of authorities into the system.

Klíčová slova

informační systém, redakční systém, databáze, samospráva, otevřená data, Flask, React, MariaDB, MongoDB

Keywords

information system, content management system, databases, local government, open data, Flask, React, MariaDB, MongoDB

Citace

KREJČÍŘ, Štěpán. *Administrační systém hlasování zastupitelstev*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Kristýna Zaklová

Administrační systém hlasování zastupitelstev

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Kristýny Zaklové. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Štěpán Krejčíř
20. května 2025

Poděkování

Rád bych poděkoval Ing. Kristýně Zaklové za všechny cenné rady, zpětnou vazbu, profesionální vedení, trpělivost a pochopení. Dále Ing. Petru Johnovi za všechny poskytnuté konzultace a ochotu nad rámec povinností. V neposlední řadě Bc. Radoslavu Kodajovi za pohotovost a zodpovědnost v situacích vyžadujících společné plánování.

Obsah

Kapitola 1

Úvod

Hlasování samosprávných orgánů na úrovni obcí i krajů je klíčovým procesem demokratického rozhodování ve veřejné správě. Mít otevřený přístup k datům z těchto hlasování může být jak v zájmu veřejnosti, tak i samotných samosprávných celků. Pro veřejnost představuje transparentnost možnost kontrolovat rozhodování svých volených zástupců a vyvíjet tlak na jejich zodpovědnost, pro samosprávný celek zase může vést k posílení důvěry veřejnosti v legitimitu rozhodovacích procesů a ve vedení samosprávy.

Zákon mnohým politickým orgánům ukládá povinnost pořizovat písemné záznamy proběhnutých zasedání a zajistit jejich veřejnou dostupnost v budově orgánu. S rozvojem otevřených dat a digitalizace veřejné správy se objevuje trend zveřejňování těchto informací online, na jehož základě je vyvíjen projekt Zastupko.cz[1] s cílem poskytovat přehledné analýzy a vizualizace hlasování zastupitelstev a dalších orgánů napříč obcemi i kraji České republiky. Projekt také otevřeně publikuje zpracovaná data ve strojově čitelných datových sadách rozdělených po jednotlivých municipalitách a funkčních obdobích, čímž může být prospěšný při rozvoji dalších podobně zaměřených nástrojů.

Hlavními cíli této diplomové práce je rozšíření administrátorského rozhraní projektu o modul umožňující manipulaci se systémem používanými databázemi, o možnost kromě momentálně podporovaných zastupitelstev spravovat také další typů orgánů, které jsou doposud upravovány skripty a přímými zásahy na databázovém serveru, a v neposlední řadě o úpravy již existujícího rozhraní reagující na aktualizované požadavky k jeho užívání. Kapitola 2 poskytuje teoretický úvod do problematiky obecních a krajských samosprávných orgánů, jejich hlasování a dostupnosti dat z těchto hlasování. Obdobně je kapitola 3 teoretickým úvodem do informačních a redakčních systémů. Nabízí také srovnání z nejpopulárnějších redakčních systémů z hlediska podílu na trhu.

Kapitola 4 shrnuje stav systému projektu Zastupko.cz před úpravami a přibližuje požadavky na rozšíření. V kapitole 5 je představen návrh rozšíření a úprav připravených na základě stanovených požadavků. Kapitola 6 popisuje konkrétní implementační detaily nejzajímavějších problému souvisejících zejména s vytvořením nové části určené k úpravám, exportům, zálohám a importům databází. Tyto zásahy byly prováděny v serverové i klientské vrstvě aplikace. V kapitole 7 je pak přiblížen postup testování implementovaných rozšíření.

[1] https://zastupko.cz

Kapitola 2

Obecní a krajské orgány

Tato kapitola popisuje základní legislativní rámec související s problematikou obecních a krajských samosprávných orgánů v České republice. U vybraných bude představen proces jejich hlasování a to, jaká data bývají z těchto hlasování dostupná. Kapitola se také dotkne způsobů, jakými vybrané orgány dávají svá data k dispozici veřejnosti.

Jako základ je třeba uvést pojem *obec*. Ta je v české legislativě [12] definována jako základní územní samosprávné společenství občanů tvořící územní celek a jako veřejnoprávní korporace s vlastním majetkem, která pečuje o rozvoj svého území, potřeby občanů a ochranu veřejného zájmu. Obce je možné dále dělit na městysy, města a statutární města dle podmínek stanovených v témže zákoně. Statutární města jsou pak dělena na městské části nebo městské obvody. Obce jsou samostatně spravovány pomocí orgánů. Mezi ty patří zastupitelstvo, rada, starosta a úřad pro obce, městysy, města, městské části a městské obvody. Výkonným orgánem může být také komise, pokud jí byl dle stejného zákona jiným orgánem v některých otázkách svěřen výkon přenesené působnosti. Orgány statutárních měst se pak skládají ze zastupitelstva, rady, primátora a magistrátu. Všechny typy obcí mohou navíc disponovat vlastními zvláštními orgány a obecní policií.

Ze zmíněného zákona je vyňato hlavní město Praha, pro něž byl místo toho přijat zákon [15], který se ale liší pouze ve věcech pro tuto práci nepodstatných. Základní definice hlavního města je shodná s předchozí definicí obce. Samosprávné orgány jsou stejné jako u statutárních měst. Praha je stejně jako statutární města taktéž dělena na městské části s vlastními orgány, jež se opět shodují s orgány městských částí a obvodů definovanými předchozím zákonem.

Důležité je představit rovněž pojem *kraj*, jenž je ukotven v zákoně [13], a je definován jako územní společenství občanů s právem na samosprávu a veřejnoprávní korporace s vlastním majetkem a zákonem vymezenými příjmy, s nimiž v rámci zákona nakládá dle vlastního rozpočtu. Také kraje jsou samostatně spravovány vlastními orgány, mezi něž podle zákona náleží zastupitelstvo, rada, hejtman, úřad, případně další zvláštní orgány konkrétních krajů.

Ze všech existujících orgánů byly pro detailnější popis vybrány zastupitelstva a rady, jelikož se jedná o klíčové rozhodovací orgány na úrovni obcí i krajů, jejichž informace z jednacích a hlasovacích procesů bývají často veřejně dostupná a poskytují nejvíce relevantních dat pro účely této práce.

2.1 Zastupitelstvo

O zastupitelstvech, zastupitelích, jejich právech a povinnostech hovoří čtvrtá hlava zákona pro obecní zastupitelstva [12], respektive čtvrtá hlava zákona pro krajská zastupitelstva [13]. Následující text vychází z těchto dvou zákonů, pokud není uvedeno jinak.

2.1.1 Obecní

Obecní zastupitelstva jsou jedním z orgánů obcí, městských částí a městských obvodů. Mohou se skládat z 5 až 55 členů. Počet je určován podle velikosti a počtu obyvatel obce nebo městské části či obvodu. Zastupitelé jsou voleni v přímých volbách plnoletými občany České republiky s trvalým pobytem v dané obci a jejich mandát v případě, že nenastanou žádné mimořádné události, trvá po celé volební období, tedy čtyři roky [16]. Všichni členové mají právo předkládat návrhy pro projednání zastupitelstvem, radou, výborům a komisím a vznášet dotazy či podněty na radu a další obecní orgány nebo jejich jednotlivce. Mezi jejich povinnosti patří zúčastňovat se zasedání zastupitelstva, plnit úkoly vyplývající z jejich funkcí a hájit zájmy občanů obce. Navíc jsou povinni dopředu hlásit, pokud by jejich účastí na konkrétním projednávání měl vznikat střet zájmů.

Zákon uděluje zastupitelstvům celou řadu pravomocí. Mezi nejvýznamnější lze zařadit například:

- schvalování programu rozvoje obce,
- schvalování rozpočtových plánů,
- vydávání obecně závazných vyhlášek,
- vypisování místních referend,
- rozhodování o nemovitostech,
- řízení o názvech ulic a dalších částí obce,
- navrhování katastrálních změn uvnitř obce,
- volení či odvolání starosty, místostarosty a dalších členů rady.

Usnesení probíhají na veřejných zasedáních zastupitelstva svolávaných typicky starostou, jejichž frekvence se odvíjí podle potřeby, ale nejméně jednou za tři měsíce. Aby bylo usnesení platné, musí být přítomna nadpoloviční většina zastupitelů. V opačném případě je zasedání ukončeno a do 21 dní nahrazeno novým. Předkládat návrhy k projednání je dovoleno zastupitelům, členům rady a výborům a o zařazení návrhů přednesených při zasedání do programu je rozhodováno zastupitelstvem.

Ze zasedání musí být do 15 dnů po jeho skončení pořízen zápis podepsaný starostou či místostarostou a ověřovateli, v němž je povinné uvádět počet přítomných členů zastupitelstva, schválený pořad jednání, průběh a výsledek hlasování a přijatá usnesení. Tento zápis je nutné uložit na obecním úřadě a zpřístupnit pro nahlédnutí.

2.1.2 Krajské

Krajská zastupitelstva mohou mít 45 až 65 členů. O jejich počtu je rozhodováno podle počtu obyvatel v kraji. Krajští zastupitelé jsou voleni pro čtyřleté volební mandáty v přímých volbách plnoletými českými občany s trvalým bydlištěm v obci patřící do území daného kraje [14]. Práva a povinnosti členů krajských zastupitelstev se podobají těm, jež platí pro zastupitele obecní. Mezi práva tedy patří předkládání návrhů a podnětů pro projednání souvisejícím orgánům nebo vznášení dotazů směřovaných na tyto orgány či jednotlivce. Také

povinnosti se příliš neliší a jsou mezi nimi mimo jiné povinnost zúčastňovat se zasedání, plnit přidělené úkoly, hájit zájmy občanů kraje a hlásit vlastní střety zájmů.

Zastupitelstva krajů mají taktéž ze zákona přisouzenu širokou škálu pravomocí. Některé z nejvýraznějších jsou například:

- předkládání návrhů zákonů Poslanecké sněmovně,
- předkládání návrhů na zrušení právních předpisů Ústavnímu soudu,
- koordinace strategií rozvoje kraje,
- schvalování koncepce rozvoje cestovního ruchu na území kraje,
- nakládání s krajským rozpočtem,
- vypisování krajských referend,
- stanovování rozsahu dopravní obsluhy na území kraje,
- volení či odvolání hejtmana, náměstka a dalších členů rady.

Způsob hlasování o usneseních krajských zastupitelstev zákon definuje podobně jako u obecních s rozdílem, že namísto starosty jsou zasedání typicky svolávána hejtmanem kraje. Ostatní podmínky jako frekvence zasedání, nutnost přítomnosti nadpoloviční většiny zastupitelů a případná náhrada se shodují. Na rozdíl od obecních zastupitelstev je předkladatelem návrhů k projednání pouze rada. Rozhodnutí o zařazení návrhů přednesených v průběhu zasedání je v kompetenci členů zastupitelstva.

Ze zasedání je nutné do 15 dnů po jeho skončení pořídit zápis podepsaný hejtmanem či jeho náměstkem a ověřovateli. Informace, které musí takový zápis povinně obsahovat, jsou shodné s podmínkami stanovenými pro zápisy z obecních zasedání. Krajské zápisy musí být dostupné k nahlédnutí na krajském úřadu.

2.2 Rada

O obecních radách hovoří čtvrtá hlava zákona [12]. O krajských radách potom analogicky čtvrtá hlava zákona [13]. Pokud není uvedeno jinak, následující odstavce čerpají informace z těchto zdrojů.

2.2.1 Obecní

Obecní rady jsou kromě obcí výkonným orgánem také v městských částech a městských obvodech. Ze své činnosti odpovídají obecním zastupitelstvům. V radách figuruje lichý počet členů od 5 do 11, který zároveň nesmí přesahovat jednu třetinu počtu členů zastupitelstva. Mezi členy patří starosta, místostarosta a další volení členové zastupitelstva obce. Pokud má zastupitelstvo méně než 15 členů, rada obce se nevolí a její pravomoci jsou delegovány na starostu.

Pravomoce obecních rad se většinou týkají řízení interních záležitostí obce. Namátkově mezi ně patří například přípravy návrhů pro jednání zastupitelstva obce a zajišťování plnění jeho přijatých usnesení, zabezpečení hospodaření obce podle schváleného rozpočtu, projednávání připomínek vznesených členy zastupitelstva či radními komisemi nebo vytváření a rušení těchto komisí.

Schůze jsou svolávány dle potřeby a na rozdíl od zastupitelstev nejsou veřejné, ale k jednotlivým bodům jednání mohou být přizváni členové zastupitelstva i jiné osoby. Mimo to pro schůze není stanoven žádný minimální počet schůzí za určitý časový interval. Ke schválení usnesení je třeba, aby nadpoloviční většina členů vyjádřila souhlas. Ze schůzí do 15 dní

od završení musí vzniknout zápisy podepsané starostou a místostarostou nebo jinými radními, jež mají obsahovat minimálně počet přítomných členů rady, schválený pořad schůze, průběh hlasování i s výsledky a přijatá usnesení. Zápisy musí být k nahlédnutí na obecním úřadě všem členům zastupitelstva obce. Zákon neukládá povinnost je poskytovat veřejnosti.

2.2.2 Krajská

Krajské rady odpovídají svou činností krajským zastupitelstvům. Skládají se z 9 nebo 11 členů, přičemž konkrétní počet je stanoven podle počtu obyvatelů daného kraje. Mezi členy rady patří hejtman, náměstek hejtmana a další členové, kteří jsou voleni ze stávajících členů zastupitelstva. Oproti obecním radám u rad krajských není stanovena žádná podmínka na minimální počet členů zastupitelstva, aby rada mohla vzniknout. Pravomoci rady může za jistých přechodných okolností vykonávat zastupitelstvo či hejtman, pokud je zastupitelstvem pověřen. Taková situace může nastat například při snížení počtu členů rady až do opětovného naplnění povinné kapacity.

Mezi některé pravomoce krajských rad patří příprava návrhů pro jednání krajského zastupitelstva a plnění přijatých usnesení, projednávání podnětů přijatých od obcí, členů zastupitelstva či komisí, zřizovat a rušit radní komise, zabezpečovat hospodaření podle schváleného rozpočtu a ukládat úkoly krajskému úřadu a kontrolovat jejich průběh. Oproti obecním radám mají navíc pravomoci zasahovat do právních jednání týkajících se dotací, odpouštění dluhů a dalších finančních záležitostí.

Pravidla pro konání schůzí krajských rad, hlasovací postup i pořizování zápisů ze schůzí se shodují s pravidly pro obecní rady. Krajské rady navíc podávají na každém zasedání krajského zastupitelstva zprávu o své činnosti.

2.3 Hlavní město Praha

Postavení hlavního města je unikátní v tom, že podle zákona [15] je považováno zároveň za obec i kraj. Tento zákon zároveň obsahuje zvláštní popis Zastupitelstva a Rady hl. m. Prahy i zastupitelstev a rad jeho městských částí, předchozí zákony se na ně tudíž nevztahují. Přesto opět obsahují spoustu paralel s určitými rozdíly, z nichž některé budou zmíněny v navazujícím textu. Detailní srovnání všech úskalí však není v kontextu této práce třeba.

Zastupitelstvo hlavního města Prahy připomíná svým uzpůsobením i pravomocemi krajská zastupitelstva. Mezi některé rozdíly patří počet členů, jichž je 55 až 70. Konkrétní číslo je stanovováno samotným zastupitelstvem. Povinnosti a práva členů jsou analogické s krajskými zastupiteli, tak jako se podobá i škála pravomocí zastupitelstva nebo postup zasedání. Zasedání svolává primátor, který k tomuto aktu je povinován i na základě písemné žádosti minimálně jedné třetiny zastupitelů. Pokud o slovo při zasedání požádá člen vlády, jeho zástupce, senátor nebo poslanec, ze zákona ho musí obdržet. Ze zasedání vzniká zápis se stejnými povinnými položkami a za stejných časových podmínek uvedených výše u krajských zastupitelstev. Zápis musí být podepsán primátorem či jeho náměstkem a ověřovateli a být k přečtení na Magistrátu města Prahy.

Zastupitelstva městských částí Prahy jsou opět tvořena jiným počet členů, který se pohybuje od 5 do 45 dle počtu obyvatel městské části. Na jejich povinnosti a práva se vztahují stejná ustanovení jako na členy Zastupitelstva hl. m. Prahy, stejně tak i průběh zasedání zasedání je obdobný. Kompetence zastupitelstva připomínají kompetence obecních zastupitelstev.

Rada hlavního města Prahy se sestává z 11 členů, mezi nimiž je primátor Prahy, náměstek primátora a další členové, u nichž zákon [15] nestanovuje, že musí být vybíráni ze členů zastupitelstva, jak je tomu u dříve představených rad. V dalších aspektech povinnosti členů i pravomoci rady analogicky zrcadlí krajské rady. Kromě toho k jejím pravomocem s určitými výjimkami přibývá možnost majetkové účasti na podnikání jiných osob, rozhodování o přijetí nebo poskytnutí úvěru i další finanční záležitosti nad rámec těch, co jsou povoleny krajským radám. Postup i povinnosti při schůzích pak zůstávají – až na detaily jako podpis zápisu z hlasování primátorem místo hejtmanem – téměř nezměněny. Rada hlavního města zastupitelstvu podává zprávu o své činnosti v šestiměsíčních intervalech.

Rady městských částí Prahy mají 5 až 9 členů, přičemž jejich počet nemůže přesahovat jednu třetinu počtu členů odpovídajícího zastupitelstva městské části, a nejsou vůbec zřizovány, pokud zastupitelstvo má méně než 15 členů. V takovémto případě jejich pravomoce přechází na daného starostu. Pokud zákon neurčí jinak, pro práva a povinnosti členů se používají stejná ustanovení jako pro Radu hl. m. Prahy, jakožto i pro pravidla schůzí a hlasování včetně pořizování zápisu. Kromě pravomocí v oblasti práce se schváleným rozpočtem mohou rady městských částí na návrh tajemníka úřadu zřizovat a rušit odbory úřadu či projednávat návrhy členů zastupitelstva a komisemi rady.

2.4 Hlasování politických orgánů České republiky

V této kapitole již bylo zmíněno, že patřičné zákony jednotlivým orgánům ukládají povinnost vytvářet z proběhnutých zasedání záznamy, které v případě zastupitelstev musí být dostupné veřejnosti a v případě rad je jejich zveřejnění ponecháno na vlastním uvážení. Ačkoliv tyto záznamy musí vždy obsahovat určité položky – jak pro zastupitelstva, tak pro rady to jsou zejména počty přítomných členů, pořad schůze, přijatá usnesení a výsledky hlasování – a další nepovinné položky jsou doporučeny Ministerstvem vnitra České republiky[1] – například jména přítomných, nepřítomných a omluvených členů –, není stanoven žádný přesný formát takového záznamu, který je tím pádem ponechán v režii obcí a krajů [26, 30, 57].

Větší obce ke správě hlasovacího procesu často využívají hlasovacích systémů. V roce 2024 práce [26, 30] zkoumaly nejčastěji používané hlasovací systémy v České republice, mezi nimiž byly identifikovány systémy Ministr[2], H.E.R. Systém[3], Usnesení.cz[4] a Konsiliář[5]. Práce [30] dále jako alternativu uvádí konferenční systémy od firmy Bosch, které jsou využívány i na mezinárodní úrovni. Všechny tyto systémy nabízí softwarová řešení zasedacích a hlasovacích procesů v případné kombinaci s hardwarovými prvky jako jsou hlasovací jednotky vybavené tlačítky pro vyjádření hlasu, čtečky čipových karet pro autentizaci členů a často také integrované mikrofony. Kromě toho mohou tyto systémy automaticky generovat zmiňované záznamy ze zasedání vyžadované českou legislativou, typicky doplněné o dodatečné, nepovinné informace. Podstatné je, že z důvodu neexistence stanovené normy se liší výstupní formáty jednotlivých systémů, a tím pádem také vhodnost těchto výstupů ke strojovému zpracování. Následující podsekce analyzují, jak jednotlivé obce a kraje v České republice svá

[1] Shrnutí povinných i doporučených položek vydané Ministerstvem vnitra ČR je dostupné na adrese https://mv.gov.cz/odk2/clanek/metodicke-materialy-k-zakonnym-zmocnenim.aspx.

[2] https://ministr.cz/

[3] https://bitest.cz/hlasovaci-system/

[4] https://usneseni.cz/

[5] https://konsiliar.hdmedia.cz/

data zpřístupňují veřejnosti a adresují pokusy o vytvoření jednotné struktury hlasovacích dat pro jejich další využitelnost.

2.4.1 Dostupnost dat z České republiky

Česká legislativa definuje otevřená data jako taková data, která jsou mimo jiné volně dostupná v otevřeném (tzn. neproprietárním), strojově čitelném formátu a evidovaná v národním katalogu [11]. V České republice se taková data stávají důležitým tématem. Ve výroční zprávě Evropské unie *Open Data Maturity Report* z roku 2024, která hodnotí usnadnění dostupnosti a znovupoužitelnosti dat z veřejného sektoru jednotlivých evropských zemí, se Česká republika umístila na 8. příčce z celkových 34[6], což značí jeden z největších meziročních nárůstů v tomto období [42]. Vzrůstající trend je patrný i z pohledu na období posledních 5 let, kdy ke konci roku 2020 byla ČR zařazena až na 21. místo. Zpráva země hodnotí podle následujících oblastí[7]:

- **Policy** – Zkoumá zásady a strategie zemí vůči otevřeným datům a to, jaká opatření země uplatňují pro jejich implementování.
- **Portal** – Měří funkcionalitu národních portálů pro otevřená data nebo to, jak dobře reagují na potřeby uživatelů.
- **Quality** – Zkoumá stav dat na národním portálu, tedy to, zda data mají náležité formáty a licence, jejich vhodnost pro strojové čtení a další aspekty.
- **Impact** – Zaměřuje se na ochotu a schopnost participujících zemí měřit znovupoužití otevřených dat a vliv tohoto znovupoužití v sociálních, ekonomických a jiných oblastech.

Naměřené hodnoty z poslední výroční zprávy ukazuje obrázek 2.1 společně s průměrem složených ze všech členských zemí Evropské unie a srovnáním s rokem 2023. Autoři zprávy připisují výrazný skok České republiky v žebříčku hlavně velkému nárůstu v kategorii Portal, v němž – jak lze také postřehnout z obrázku – doposud daleko zaostávala za průměrem. Vyzdvižené jsou zejména snahy přizpůsobit národní web pro otevřená data nazvaný Portál o datech[8] na základě přímých rozhovorů s testovacími skupinami uživatelů a dotazníkového šetření uživatelských zkušeností.

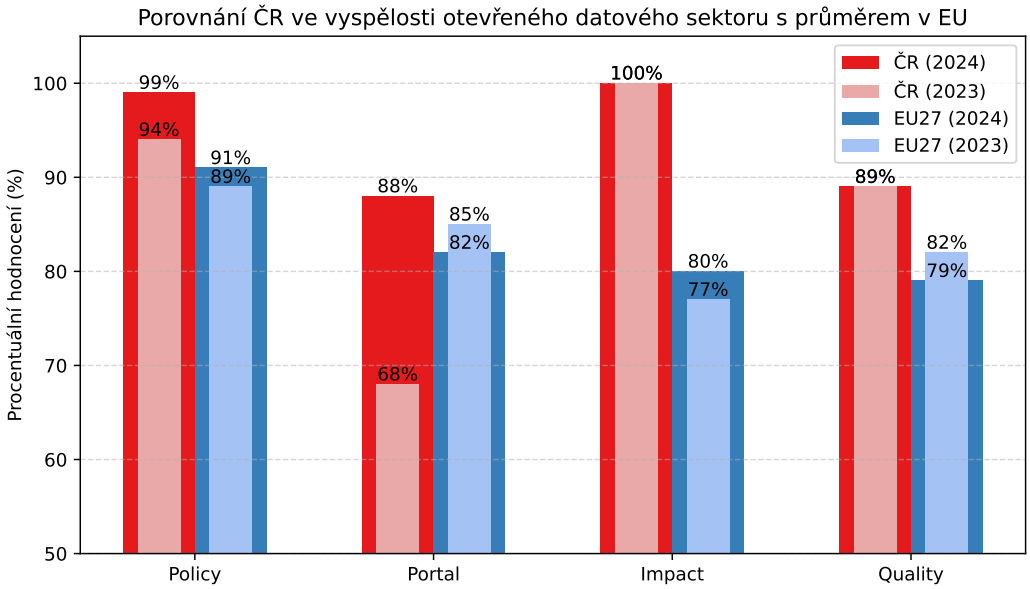
I přes vysoké postavení České republiky v sektoru otevřených dat však situace s dostupností dat ze zasedání politických orgánů není ideální. Detailním průzkumem těchto datových sad v Národním katalogu otevřených dat (NKOD)[9] se zabývaly práce [26, 30, 57]. Od posledního průzkumu byly kompletně odstraněny datasety z hlasování Zastupitelstva města Děčín a městské části Prahy 6. Naopak přibyla průběžně aktualizovaná datová sada z hlasování Zastupitelstva Plzeňského kraje ve formátu XML. Jedná se o první kraj publikující svá hlasovací data v NKOD. Dostupnost a aktualita dalších datasetů se shodují s průzkumy z roku 2024 [26, 30]. Nadále je platný také zmíněnými pracemi ověřený fakt, že formát a struktura napříč datasety jsou často výrazně odlišné. Velké množství obcí místo

[6] Zahrnuty jsou nejen členské státy Evropské unie, ale také kandidátské země a státy Evropského sdružení volného obchodu, které se rozhodly zúčastnit.

[7] Názvy oblastí byly ponechány v angličtině, jelikož jsou takto uvedeny i na Portálu o datech, viz například https://data.gov.cz/2024/12/16/%C4%8Desko-v-top-10-v-hodnocen%C3%AD-ODMR.html.

[8] Dostupný na https://data.gov.cz/.

[9] Národní katalog otevřených dat je součástí Portálu o datech.



Obrázek 2.1: Srovnání České republiky s průměrem zemí Evropské unie v oblastech hodnocenými výroční zprávou Open Data Maturity Report. Světlé sloupce slouží k ilustraci meziroční změny. Vertikální osa začíná na 50 procentech. Hodnoty za rok 2024 převzaty z [42], hodnoty za rok 2023 z https://doi.org/10.2830/384422.

do NKOD publikuje určité relevantní informace na své vlastní webové stránky. Tyto informace však často nejsou poskytovány jako ucelené datasety, ale spíše roztříštěné výstupy, mnohdy navíc ve formátech nevhodných pro strojové zpracování (např. PDF). I v tomto ohledu ve srovnání s průzkumy z roku 2024 zůstává situace z hlediska jednotnosti spíše nepříznivá. Ať už tedy jde o data dostupná skrze NKOD, nebo přes webové stránky měst, konverze do jednotného modelu by vyžadovala proces různě obtížného předzpracování.

2.4.2 Datové modely hlasování

Větší transparentnosti institucí s rozhodovacími pravomocemi by mohlo přijít ku prospěchu, pokud by v České republice ve světle snah vyvíjených ke zvýšení dostupnosti otevřených dat byla zavedena jednotná forma zápisu dat ze zasedání politických orgánů. V české otevřené datové infrastruktuře existují takzvané otevřené formální normy (OFN)[10], tedy doporučení pro vybrané typy datových sad, jež mají za cíl zajištění interoperability a jednodušší využitelnosti bez závislosti na tom, od jakého zdroje data pochází. Přestože v minulosti alespoň jednou proběhla snaha o vytvoření OFN pro hlasování zastupitelstev obcí [59], doposud nebyla tato ani žádná jí podobná finalizována a uvedena do praktického použití.

Na mezinárodní úrovni existuje projekt *Popolo*[11], jenž představuje robustní standard k popisování politických dat, mezi něž patří namátkově data organizací, osob a jejich členství, schůzí, hlasování a dalších položek. Teze [58] zmiňuje komplexnost tohoto modelu jako jednu z možných překážek při implementaci a potenciální důvod vzniku řady dalších více či méně specifických modelů, které byly ve světě navrženy. Jedním z výrazných

[10] Oficiální přehled všech OFN je na stránce https://data.gov.cz/ofn/.
[11] https://popoloproject.com/

uváděných národních modelů je projekt *Open Parl Data*[12] vznikající za účelem přinést pomocí standardního formátu a volně dostupného rozhraní větší transparentnost do hlasování švýcarského parlamentu. Ve světě ani v České republice se přes tyto i jiné pokusy doposud nedostalo oficiální adopce žádnému otevřenému modelu pro zaznamenávání zasedání a výstupů hlasování.

Teze [58] poté navrhuje vlastní generický model[13] ve formátu JSON. Ten rozděluje ukládané informace do čtyř logických celků – základní informace o samosprávě, politické subjekty, zastupitelé a data o jednotlivých volebních obdobích, zasedáních a hlasováních – a byl navržen tak, aby pokryl různé varianty záznamů hlasování včetně anonymních i jmenovitých. V tezi je však zmiňována potřeba dalšího vývoje k dosažení větší robustnosti a jako následující cíl je uváděno zavedení způsobu ukládání informací o podpůrných dokumentech a materiálech přidružených k daným hlasováním. Chystané změny mají potenciál rozšířit adopci modelu do dalších projektů v České republice i v zahraničí.

[12] https://opendata.ch/projects/openparldata/
[13] https://github.com/zastupko/data-model

Kapitola 3

Informační a redakční systémy

Tato kapitola představí různé pohledy na definici informačních systémů a popis základních komponent, z nichž se skládají. Sumarizuje také nejčastěji používané architektury či v současnosti nejoblíbenější vývojové nástroje pro jednotlivé z komponent. Dále se kapitola věnuje redakčním systémům. Největší důraz je s ohledem na téma této práce kladen na řešení v podobě webových aplikací.

3.1 Informační systémy

Na termín *informační systém* (anglicky *information system*, zkráceně *IS*) je možné nahlížet několika v zásadě podobnými způsoby s určitými odchylkami. Mnoho z nich používá pojmy *data* a *informace*, které jsou s informačními systémy úzce spjaty. Daty se myslí základní popisy věcí, událostí, aktivit a transakcí z domény daného IS. Data nebývají organizována tak, aby sama o sobě měla pro uživatele význam. Informací se pak rozumí data, která byla zpracována či vložena do kontextu tak, aby významu nabyla [43, 52]. Data tedy mohou být například produkty a zákaznická hodnocení. Přiřazením hodnocení ke konkrétním produktům a spočítáním průměrného hodnocení produktu je získána informace.

Ani jeden z doposud představených pojmů však nemusí být nutně spjat s informačními technologiemi a počítači. Podobně je to potom s definicí samotného informačního systému, jak říká zdroj [39]. Jako příklad uvádí italské vinařství Barone Ricasoli, které dle něj už koncem 19. století muselo mít informační systém, jenž firmě dovoloval sledovat objednávky, platby, skladové zásoby a další důležité informace. IS tedy definuje jako formální, sociotechnický, organizační systém určený k práci s daty, která zahrnuje jejich sběr, zpracování, schraňování a distribuci.

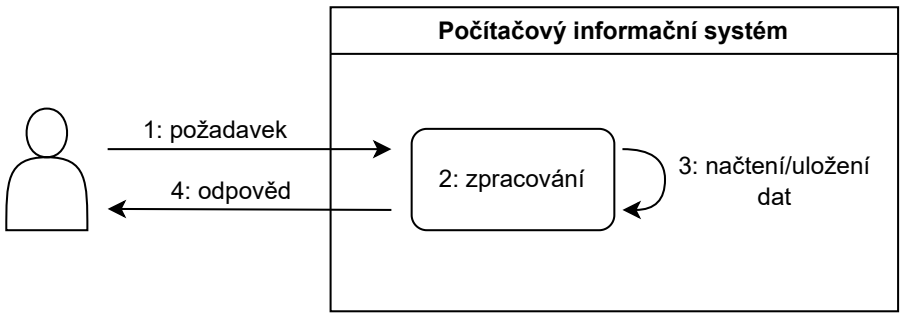
Definice zdroje [4] je prakticky identická v tom, že termín popisuje jako systém k práci s daty zahrnující operace shodné s předchozím pohledem na věc. Navíc ale definici rozšiřuje o poznatek, že tyto operace jsou prováděny v rámci jednoho podniku či mezi podnikem a jeho okolím.

Od předchozích definic se mírně liší zdroj [43], jenž na termín hledí jako na systém pro práci s daty, jehož hlavním účelem je včas dodávat správné informace k lidem, k nimž se dostat mají, ve vhodném množství a formátu. Zdroj dále podporuje tvrzení, že ne všechny informační systémy jsou spojeny s počítači. Většina z nich ale ano, z čehož důvodu se spojení informační systém v dnešní době často používá jako synonymum pro *počítačový informační*

systém (anglicky *computer-based information system*[1], zkráceně *CBIS*), který popisuje jako IS využívající počítačové technologie k plnění alespoň některých ze svých funkcí.

Zdroj [52] zahrnuje počítače už ve své základní definici, když říká, že IS je kombinací informačních technologií a aktivit lidí, již tuto technologii na různých úrovních organizace používají k managementu, podpoře podnikových procesů nebo jako pomoc při rozhodování.

Z předchozích definic lze tedy dojít k závěru, že IS je poměrně široký pojem, který v sobě může zahrnovat řadu složek od hardwaru a softwaru přes data a databáze až po osoby, které systém obsluhují, a činnosti, jež vykonávají. Pro účely této práce postačí o něm přemýšlet jako o softwaru pracujícím s databází, jejíž data umožňuje jeho uživatelům zobrazovat a případně spravovat (viz obrázek 3.1).



Obrázek 3.1: Vizualizace zjednodušené definice počítačového informačního systému. Uživatel systému odešle požadavek, který systém zpracuje a uživateli vrátí odpověď. Třetím krokem se myslí načtení či uložení dat do úložiště. Uložení nemusí být provedeno v případě, že je přijat požadavek, jenž data nijak neupravuje, nebo pokud je požadavek na úpravu dat z jakéhokoliv důvodu zamítnut. Inspirováno [39].

3.2 Architektura počítačových informačních systémů

V předchozí podkapitole byl definován způsob, jímž je v této práci nahlíženo na počítačový informační systém. Nyní je potřeba vysvětlit, z jakých komponent se takový systém skládá a jakou tyto komponenty dohromady tvoří architekturu. Zdrojem pro tuto sekci je [17], pokud není uvedeno jinak.

Systémová architektura určuje entity, jež spolu v systému budou komunikovat a jakým způsobem to budou provádět, jaké jsou jejich role a povinnosti, a také jak se tyto entity promítnou do fyzické infrastruktury. Pro počítačové informační systémy a distribuované aplikace obecně je nejzásadnější a nejběžnější architekturou architektura nazývaná *klient-server* [17, 32]. Ta se typicky dále dělí na *dvouúrovňovou architekturu* (angl. *two-tier architecture*) a *tříúrovňovou architekturu* (angl. *three-tier architecture*). V obou případech jsou základními komponentami *server*, *klient* a *databáze*, jejichž obecné role jsou popsány níže v této podsekci.

U obou variant zmíněné architektury by měla aplikace disponovat prezentační logikou, která dohlíží na interakci s uživatelem, aplikační logikou, jejímž úkolem je starat se o procesy specifické pro danou aplikaci, a datovou logikou poskytující úložiště pro data. Dvouúrovňová varianta architektury všechny tyto úlohy dělí mezi klienta a server s datovou vrstvou typicky

[1] V některých textech také uváděn pouze jako *computer information system*.

sídlící přímo na serveru. U tříúrovňové varianty třetí úroveň představuje samostatný od aplikačního serveru oddělený databázový server, činící z aplikačního serveru prostředníka mezi klientem a daty. Takovéto oddělení může umožnit lepší škálovatelnost a separovaný vývoj. Nevýhodou je potenciálně větší komplexnost aplikace a větší nároky na síť z důvodu nárůstu počtu oddělených komponent, co se spolu potřebují dorozumívat.

Klient

Klient je program, jenž posílá požadavky na server a zpracovává obdržené odpovědi (serverové programy jsou přiblíženy níže). Sám klient tedy nemusí sloužit pouze k zobrazování obdržených odpovědí, ale jeho úkolem může být také provádění výpočtů, jimiž by bylo zbytečné nebo neefektivní zatěžovat server [5, 17]. V dnešní době mohou klientské programy běžet jak na osobních počítačích, tak na chytrých telefonech [31]. U webových aplikací roli klienta zastává internetový prohlížeč, zatímco u nativních aplikací je klientem software instalovaný přímo na zařízení uživatele.

Z pohledu informačních systémů je klientem aktivní program nabízející uživateli ovládací rozhraní k zobrazování a správě dat, který na základě uživatelských požadavků iniciuje komunikaci se serverovou částí aplikace a odpověď serveru prezentuje zpět uživateli v člověkem čitelné formě.

Server

Obecně lze server definovat jako program nebo skupinu programů běžících na zařízení v síti, jejichž úkolem je přijímat a obsluhovat požadavky klientů, případně manipulovat s dostupnými zdroji [5, 17]. Serverové programy v praxi nejčastěji běží na výkonných strojích ve větších datových centrech [31].

V kontextu počítačových informačních systémů se jedná o pasivní program čekající na požadavky klientů. Jakmile požadavek obdrží, zpracuje jej a klientovi zpřístupní požadovaná data či informace nebo dle něj manipuluje s již existujícími daty. Server také může podporovat ověřování správnosti či autorizaci požadavků, tedy mimo jiné odmítat neznámé požadavky a zabránit neoprávněným přístupům a manipulacím s daty.

Databáze

Databáze slouží podnikům k ukládání dat, jež popisují entity z oblasti působnosti daného podniku a vztahy mezi nimi. Takto uložená data nejsou nijak závislá na aplikacích, které k nim přistupují [32]. Data jsou obvykle systematicky strukturovaná a roztříděná do k sobě logicky pasujících celků [43, 52]. V závislosti na typu aplikace může být využíváno centrální nebo distribuované databáze. Druhý uvedený způsob využívá rozdělení databáze na menší části umístěné na potenciálně různých fyzických místech, což může poskytnout rychlejší přístup k datům z důvodu fyzické blízkosti uživateli a schopnost ponechat systém funkční při výpadku jednoho uzlu, ale vnáší nové problémy jako nutnost zajišťování konzistence dat mezi jednotlivými uzly v případě datových závislostí [32, 51, 52].

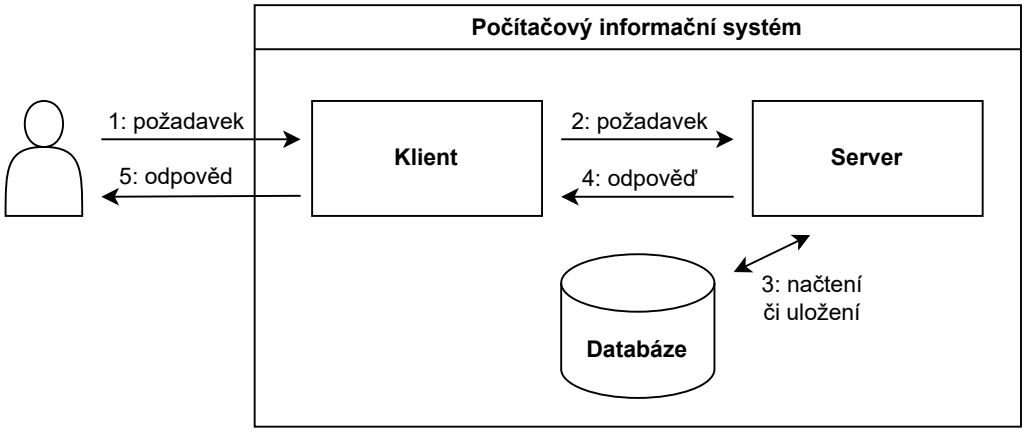
Databáze můžou mít různé typy, do nichž se dělí podle způsobu ukládání a práce s daty. Literatura zaměřená na informační systémy nejčastější zmiňuje relační databáze a NoSQL databáze [32, 39, 52], z čehož lze usoudit, že tyto dva typy jsou nejběžnějším prostředkem pro uchovávání dat, s nimiž v oblasti počítačových informačních systémů server pracuje. Uvedené zdroje popisují relační databáze jako takové, jež ukládají data do dvoudimenzionálních tabulek se schématem předem definovanou strukturou, kde každý řádek repre-

zentuje datovou entitu a každý sloupec atribut entity, který může nabývat hodnot pouze z předem definované domény (entita může být například produkt a jeden z atributů jeho cena). Naproti tomu NoSQL databáze poskytují flexibilnější možnosti ukládání dat, jelikož nevyžadují předem definované schéma, což může usnadňovat práci s měnícími se nebo nestrukturovanými daty. Existují také další, pro IS méně dominantní typy, mezi něž patří mimo jiné temporální, prostorové nebo objektově orientované databáze [20].

Shrnutí

Představeny byly části, ze kterých se skládá běžný počítačový informační systém. Klient zprostředkovává uživatelské rozhraní a odesílá zadané požadavky, server tyto požadavky zpracovává, vrací odpovědi a zajišťuje běh aplikační logiky, zatímco databáze poskytuje úložiště pro data, nad nimiž jsou tyto operace prováděny.

Na základě této architektury je možné dále upřesnit obrázek 3.1 jejím zakomponováním do diagramu. Ve výsledném obrázku 3.2 je vyobrazen CBIS využívající tříúrovňové varianty architektury klient-server. Oproti původnímu obrázku byl obecný krok *zpracování* rozdělen na položky *server* a *klient*, čímž diagram blíže popisuje typickou realitu.



Obrázek 3.2: Rozšířená vizualizace schématu počítačového informačního systému běžícího na tříúrovňové architektuře klient-server. Uživatel zadá požadavek klientovi, který jej odešle na server. Server načte potřebná data nebo zapíše nová, pokud o to byl požádán, a zformuluje odpověď pro klienta, jenž ji zobrazí uživateli. Komponenty *server*, *klient* a *databáze* jsou popsány v kapitole 3.2.

3.3 Komunikace mezi komponentami

Sekce 3.2 se zmiňuje o skutečnosti, že jednotlivé komponenty počítačového informačního systému si mezi sebou posílají zprávy. V návaznosti na ni tato sekce říká, jakým způsobem k takové komunikaci může docházet.

Jakmile existují systémové komponenty, které jsou od sebe vzájemně odděleny, vzniká potřeba ustanovení způsobu komunikace. Toto ustanovení je definováno pomocí *aplikačního rozhraní* (anglicky *application programming interface*, zkráceně *API*), a jeho cílem je specifikace možností komunikace, jíž se musí obě strany držet, aby mohlo docházet ke sdílení dat a metod mezi danými komponentami [8, 49]. Poskytnutím API je navíc zajištěno,

že komponenty mají přístup jen k těm částem svého protějšku, které protějšek sám v rámci rozhraní otevře, což systému může dodat jednu vrstvu zabezpečení [49].

Vzhledem k zaměření této práce cílí tato sekce především na taková API paradigmata, která bývají nejčastěji používána při vývoji počítačových informačních systémů. Nutno podotknout, že způsobů, jakými lze implementovat rozhraní mezi danými komponentami, je více, než je uvedeno níže. Vhodnost jejich použití se může různit v závislosti na zaměření aplikace.

3.3.1 Rozhraní klient/server

Aplikační rozhraní mezi klientem a serverem počítačového informačního systému definuje klientovi způsob, jakým převádí požadavky uživatele na zprávy srozumitelné serveru, a kam tyto zprávy zasílat, aby je server přijal. Obdobně určuje serveru, jak naslouchat zprávám klientů a jak na ně reagovat. Jinými slovy definuje pravidla pro komunikaci, která klienti využívají, aby serveru předali uživatelské požadavky ke čtení a správě dat v IS, a server podle nich vrátil klientům odpovědi o výsledcích těchto operací [49].

Přístupů, jimiž lze přistupovat k návrhu rozhraní přistupovat, existuje mnoho, z nichž každý má specifické výhody a nevýhody v závislosti na požadavcích aplikace a prostředí. Jednoznačně nejpoužívanějším přístupem je REST [8, 27]. Mezi další často využívané patří mimo jiné GraphQL a gRPC [27].

REST

Aplikační rozhraní po vzoru REST (Representational State Transfer) funguje na principu zpřístupňování *zdrojů* (anglicky *resources*). Za zdroj lze považovat jakoukoliv entitu patřící do domény systému, jejíž data si přejeme potenciálně zpřístupnit vnějším programům [23, 46]. V informačním systému spravujícím produkty a zákaznické recenze by zdrojem tedy mohl být například produkt, zákazník nebo recenze. Každý zdroj má typicky zpřístupněnou vlastní URL (tzv. endpoint nebo koncový bod), na niž klient může serveru posílat požadavky související s daným zdrojem [23, 27].

Pro přenos požadavků se nejčastěji používá protokol HTTP [8, 23], jenž nabízí běžně používané metody jako `GET`, `POST`, `PUT` a další[2], pomocí nichž je snadné v API definovat CRUD operace[3]. Jeden endpoint je schopen rozeznávat více metod a tedy i současně obsluhovat více případů užití souvisejících se stejným zdrojem. Pokud by uživatel chtěl načíst seznam všech produktů, klient by za něj poslal HTTP požadavek s metodou `GET` na hypotetický koncový bod `/products`. V případě, že by se jednalo o validní požadavek, by server v odpovědi kromě stavového kódu HTTP[4] poslal také aktuální seznam produktů. Ten může být přiložen v těle odpovědi ve formátu XML nebo JSON[5] [23, 27]. Tyto formáty lze posílat i v samotném požadavku, pokud klient potřebuje serveru předat data. Pro situaci, kdy by chtěl uživatel přidat nový produkt, klient odešle požadavek s metodou `POST` na stejný hypotetický endpoint `/products`.

[2] HTTP metody používané ke specifikaci účelu HTTP požadavku. Jejich seznam je k nahlédnutí na https://datatracker.ietf.org/doc/html/rfc2616#section-9.

[3] CRUD je akronym pro *Create, Read, Update, Delete*. Jedná se tedy o operace k vytváření, čtení, aktualizaci a mazání zdrojů.

[4] Stavové kódy HTTP přicházejí v odpovědi HTTP serveru a informují o úspěchu, neúspěchu či dalších podrobnostech týkajících se původního HTTP požadavku. Jejich seznam je k nalezení na https://datatracker.ietf.org/doc/html/rfc2616#section-10.

[5] XML a JSON jsou formáty mimo jiné využívané k serializaci dat.

REST byl navržen jako bezestavové rozhraní [22], což znamená, že server nevyužívá žádné informace z předešlých požadavků a každý z nich vykonává izolovaně. Z toho rovněž plyne, že požadavek musí obsahovat veškeré informace potřebné k jeho vykonání. Pokud aplikace vyžaduje udržování stavu, náleží tato povinnost klientovi.

Jednou z potenciálních nevýhod principu REST je fakt, že klient nemá kontrolu v tom, jaká data server posílá zpět. Pokud v představeném hypotetickém scénáři klient pošle `GET` požadavek na endpoint `/products`, v čisté implementaci RESTu nemá čím specifikovat, jaké konkrétní údaje si přeje dostat zpět. Pokud by klient potřeboval pouze jména všech produktů, ale server je naprogramován, aby odeslal i detailní informace o produktech, je síť zatěžována zbytečnými daty navíc [7, 35].

GraphQL

GraphQL je dotazovací jazyk a běhové prostředí pro implementaci a provoz aplikačního rozhraní navržený a vyvíjený jako open-source společností Meta z frustrace s tehdy existujícími architekturami aplikačních rozhraní [10]. Na data nahlíží jako na graf, v němž datové entity představují uzly a vztahy mezi nimi jsou hranami. Využije-li se příklad z podsekce o principu REST, uzel by mohl být produkt, zákazník či recenze. Vztah mezi produktem a recenzí by mohl reprezentovat jednu z hran.

Při používání GraphQL je nutné na serveru definovat schémata popisující entity, jež mají být poskytnuty aplikačním rozhraním, a dotazy či požadavky, jež dovolí klientům nad entitami provádět CRUD operace. Na rozdíl od principu REST server naslouchá na jediném endpointu [27], který není spojen s žádným konkrétním zdrojem. Toto je možné díky způsobu, jakým bylo v GraphQL implementováno dotazování, kdy je požadovaný typ, tedy GraphQL ekvivalent zdrojů z REST, zmíněn přímo v těle požadavku [7, 8, 40]. Podobně jako při principu REST nejsou ani požadavky GraphQL závislé na konkrétním transportním protokolu, ale typicky bývá využíváno protokolu HTTP [40]. GraphQL požadavek je řetězec se syntaxí odpovídající dotazovacímu jazyku GraphQL, jenž je klientem připojen do těla odesílaného HTTP požadavku, zatímco server do těla odpovědi klientovi vkládá data ve formátu JSON [7].

GraphQL je schopen řešit jeden z problémů principu REST vzpomínaný v předchozí podsekci, kdy by si klient vystačil pouze se jmény produktů v databázi, ale server umí odpovědět pouze kolekcí obsahující i další informace o produktech navíc. Klient má možnost pomocí dotazovacího jazyka GraphQL ve svém požadavku popsat, jaké informace si přeje od serveru získat, a to nejen v rámci jediného typu. V jednom GraphQL požadavku je možné zaslat více dotazů napříč typy definovanými serverovým schématem a pro každý typ specifikovat konkrétní požadovaná pole [7, 40], tedy pro příklad pouze samotné produktové jméno, což serveru umožňuje upravovat obsah odpovědí dle přání klienta a nezatěžovat síť přenosem redundantních informací.

GraphQL nepřináší pouze výhody. Z povahy, jakým funguje jeho dotazování, má zvýšené nároky na výkon serveru, jenž musí nejprve parsovat mnohdy komplexní požadavky [7, 27]. S tím souvisí to, že se server může stát náchylnější na útoky typu DoS[6], které komplexních požadavků můžou využít k přetížení systému. Proměnlivost dotazů zase může být problematická při jejich přednačítání z mezipaměti [7] [7].

[6] Útoky DoS (denial of service) stojí na principu zahlcení stroje či služby nesmyslnými požadavky s cílem přetížit systém a znemožnit nebo zpomalit jeho běžnou činnost.

[7] Přednačítáním z mezipaměti je myšlen anglický pojem *caching*, tedy proces ukládání nedávných dotazů a odpovědí. Při stejném dotazu je možné použít odpověď z cache místo opětovného posílání dotazu na server.

gRPC

Framework gRPC od společnosti Google je open-source variantou paradigmatu RPC (Remote Procedure Call) [8, 25], které klientům umožňuje napřímo volat serverové metody a předávat jim argumenty tak, jako by tyto funkce byly lokálně implementovány v samotném klientovi [27]. Framework od Googlu k definici klientům přístupných datových struktur a metod používá tzv. *protokolové buffery* (zkracovány jako *protobuf*). Překladač z daného protobufu vygeneruje kód pro požadovaný programovací jazyk, který je poté integrován do serverových a klientských programů k obsluze komunikace [8, 25]. gRPC je nezávislý na platformě a dovoluje, aby jednotliví komunikující klienti a servery běželi na odlišných programovacích jazycích [25].

Na rozdíl od aplikačních rozhraní implementujících REST nebo GraphQL je framework gRPC závislý na konktrétním transportním protokolu. Používá novější verzi protokolu HTTP zvanou HTTP/2. Ta gRPC umožňuje využívat binární serializace dat s kompresí [25, 28]. Když klient zavolá vzdálenou metodu, podle protobufu je vytvořena binární zpráva o jejím volání a argumentech, která je protokolem HTTP/2 doručena serveru, jenž zprávu zpracuje a provede požadovanou metodu. Odpověď serveru je opět přenesena v binární zprávě protokolem HTTP/2 [25].

Použití druhé verze protokolu HTTP zajišťuje vyšší efektivitu přenosu dat po síti oproti starší verzi HTTP/1.1, jež je běžně používána pro REST a GraphQL. To může být užitečné zejména při přenášení velkých objemů [28, 35]. Mezi další výhody HTTP/2 patří podpora multiplexingu[8] nebo streamování dat[9], čehož gRPC využívá a dovoluje do aplikačního rozhraní zavést podporu streamování na obě strany, tedy jak od serveru ke klientovi, tak i druhým směrem. Mezi některé nevýhody frameworku gRPC podle zdroje [25] patří skutečnost, že při drastických změnách datového schématu či zpřístupněných metod a tím pádem velkých zásazích do protobufu je třeba vygenerovat nový kód pro servery i klienty, což může vývojový proces činit složitějším. Zdroj dále uvádí, že framework nemusí být vhodný pro aplikační rozhraní otevřená veřejnosti, protože pro klienty nenabízí tolik kontroly nad požadavky jako například GraphQL nebo typovou flexibilitu odpovědí ve formátu JSON u GraphQL i REST.

3.3.2 Rozhraní server/databáze

Aby mohl server obsluhovat požadavky klientů, potřebuje přístup k datům, nad nimiž informační systém pracuje. Mezi serverem a databází tedy musí existovat rozhraní také zvané *driver* nebo *connector*, jež definuje, jak mezi sebou tyto komponenty komunikují. Jinými slovy driver převezme požadavek serveru na čtení či úpravu dat a přetransformuje ho do podoby, které rozumí daná databáze. Obdobnou cestou poté odpověď z databáze přeloží serveru [48].

Na rozdíl od rozhraní mezi klientem a serverem představených v předchozí podsekci, která jsou z velké části nezávislá na použitých programovacích jazycích, je rozhraní mezi serverem a databází často více či méně zakotveno tím, na jakém konkrétním jazyku server běží a s jakým databázovým systémem spolupracuje. Některé databáze poskytují vývojářům vlastní proprietární implementace rozhraní pro konkrétní jazyky [1], u dalších je využíváno komunitních řešení. Takových implementací je velké množství pro různé kombinace programovacích jazyků a databázových systémů. Jako příklad se dají uvést různé

[8] Multiplexing je kombinace několika signálů do jednoho přenosového kanálu. V případě HTTP/2 se jedná o přenos více HTTP požadavků či odpovědí v jednom spojení.

[9] Streamování dat znamená přenášení nebo přijímání dat v nepřetržitém toku.

knihovny umožňující aplikacím komunikovat s databázovým systémem MySQL. Knihovna `mysqlclient` implementuje aplikační rozhraní pro aplikace psané v programovacím jazyce Python. Knihovna `mysqli` pak slouží k podobnému účelu aplikacím napsaným v jazyce PHP.

Mimo tyto implementace vázané na technologie existují také rozhraní JDBC a ODBC, jejichž zásadní vlastností je, že nabízí jednotné rozhraní pro běžně používané databáze [21, 36], z čehož plyne, že při jejich použití není třeba přepisovat kód aplikace v případě migrace na nový databázový systém. Některé studie ovšem naznačují, že je tato výhoda vykoupena nižším výkonem [1].

3.4 Redakční systémy

Před představením redakčních systémů je třeba ujasnit související pojem *obsah* (angl. *content*), který sám o sobě nemusí být nutně tak přímočarý, jak může působit na první pohled. Zdroj [6] jeho definování věnuje několik kapitol celkově o téměř 60 stranách. Není ovšem v možnostech této práce podrobně obsáhnout všechny tyto myšlenky. Stručně lze tento pojem shrnout jako informaci, která byla opatřena daty (v tomto případě také zvaná metadata), formátem a strukturou, aby bylo umožněno její strojovou organizaci, správu a publikaci [6] (pojmy *data* a *informace* byly představeny v sekci 3.1). Je nejen tvořen, ale také určen ke konzumaci lidmi [2]. V reálné situaci se může jednat například o zpravodajský článek doplněný o jméno autora a štítky klasifikující jeho obsah do patřičných kategorií.

Samotné redakční systémy (anglicky *content management systems, zkráceně CMS*)[10] mají největší uplatnění při správě webových stránek a aplikací [2, 6]. Před jejich příchodem dominovaly statické weby. Blogový článek například typicky existoval jako samostatný HTML soubor. Obdobně jakákoliv stránka patřící pod daný web. Se zvětšujícím se množstvím obsahu bylo obtížné si o něm držet přehled a jeho správa se stávala složitější. Redakční systémy začaly vznikat jako odpověď na tento problém [2].

Podobně jako informační systém v sekci 3.1 je lze popsat vícero způsoby, jež se od sebe mohou více či méně lišit. Zdroj [6] pojem redakční systém definuje jako nástroj k práci s obsahem. Tu dále dělí na sběr, správu a publikování. *Sběrem* je myšleno nabytí informací ze zdroje, jejím převedením do požadovaného formátu a přidání metadat. *Správou* pak ukládání obsahu do repozitářů či databází ve stanovené struktuře a práci s administrativními daty, jako můžou být například data o uživatelích systému. *Publikováním* se myslí načtení obsahu a jeho zpřístupnění cílovému publiku. Zjednodušení všech zmíněných operací by mělo být fundamentální funkcí všech redakčních systémů.

Zdroj [2] na redakční systémy nahlíží jako na softwarový balík určený k částečnému automatizování práce spojené s efektivní správou obsahu, mezi niž zahrnuje vytváření obsahu a jeho úpravy a zpřístupnění konzumentům, což je přístup velmi podobný předchozímu zmiňovanému. Zdroj [2] ovšem dále doplňuje, že CMS většinou běží na serveru a zdůrazňuje, že se nejedná o monolitický systém, nýbrž o spojení velkého množství potenciálně autonommě pracujících částí, jež z pohledu uživatele tvoří celek. Jako příklad některých z těchto částí uvádí rozhraní k editaci obsahu, datová úložiště nebo publikační mechanismy.

Náhled zdroje [55] se od dvou výše zmiňovaných liší tím, že redakční systém nevnímá jako software, ale jednoduše jako samotný způsob, kterým je obsah spravován během celého svého cyklu od vytvoření až k publikaci. Software, jehož cílem je pomoci tento systém dodržovat, potom nazývá redakční *software* (anglicky content management software). Pouhé

[10] Redakční systémy někdy bývají v češtině nazývány také *systémy pro správu obsahu*.

nainstalování takového softwaru firmě ale nemá umožnit účinnou správu obsahu, pokud není zároveň využíváno systému. Vedle toho zdroj [55] uvádí, jakou funkcionalitu by měl redakční software nabízet, a to mimo jiné tvorbu a revizi obsahu, verzování obsahu, přidávání metadat, publikování a administrativní funkce, což se víceméně shoduje s nároky, jaké předchozí zdroje kladly na redakční *systém*.

Z předchozích odstavců vyplývá, že redakční systémy lze vnímat jako jednu aplikaci, softwarový balík s potenciálně decentralizovanými částmi nebo pouhou metodiku správy obsahu, byť základní principy zůstávají stejné a ve všech případech je patrná především snaha přivést efektivní vytváření, organizaci a publikaci obsahu. Pro účely této práce bude na redakční systém nahlíženo spíše z pohledu zdrojů [2, 6], tedy jako na aplikaci nebo sadu aplikací, jejichž účelem je umožnit uživatelům zmiňované operace s obsahem. Ideálně by systém měl tuto funkcionalitu uživatelům nabízet bez nutnosti, aby disponovali hlubšími technickými znalostmi.

3.5 Dělení redakčních systémů

Redakční systémy lze rozdělit podle různých kritérií. Níže je popsáno dělení dle účelu, tedy hlavního funkčního zaměření daného systému, a architektury. Toto dělení bylo vybráno protože dobře demonstruje vhodnost jednotlivých řešení pro specifické potřeby koncových uživatelů.

Dělení podle účelu, tedy podle primárního způsobu použití daného systému, je možné podle zdroje [2] učinit na čtyři nejvýznamnější skupiny, mezi nimiž však může v konkrétních případech nastávat překryv a v závislosti na potřebách mohou existovat systémy vykazující znaky spadající pod několik následujících kategorií současně. Jedná se tedy spíše o teoretický přehled:

- **Správa webového obsahu** *(Web content management, WCM)* – Tyto redakční systémy jsou zamýšleny zejména pro časté doručování obsahu konzumentům webové stránky. Konzumentem v tomto případě typicky bývá všeobecná veřejnost. Webové CMS bývají postaveny na silných možnostech oddělení obsahu od jeho prezentace.

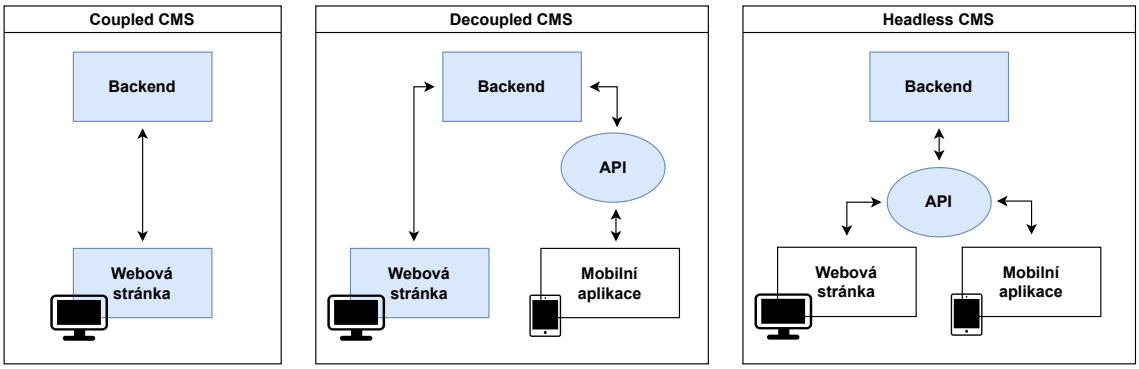
- **Správa podnikového obsahu** *(Enterprise content management, ECM)* – Hlavním cílem systémů tohoto typu bávý správa obecného obchodního obsahu pro použití v rámci daného podniku, jenž systému využívá. Mezi obchodní obsah se můžou řadit například životopisy zájemců o pozici nebo záznamy o úrazech. Tyto systémy většinou vynikají v možnostech kolaborace, řízení přístupu a manipulace se soubory.

- **Správa digitálních aktiv** *(Digital asset management, DAM)* – Systémy pro správu digitálních aktiv slouží k manipulaci se soubory, a to převážně s obrázky či jinou grafikou, videem, zvukovými nahrávkami a podobným obsahem. Klíčovými schopnostmi těchto systémů by měla být správa metadat a funkce pro jednodušší a případně hromadné úpravy a revize digitálního obsahu.

- **Správa záznamů** *(Records management, RM)* – Tato kategorie zahrnuje systémy pro manipulaci s transakcemi a dalšími obchodními záznamy, mezi něž se řadí například smlouvy nebo evidence tržeb. Takovéto systémy by měly disponovat přehlednými a efektivními možnostmi prohledávání historie záznamů.

Dělení podle architektury se týká způsobu, jakým komunikuje vrstva operující s daty (backend) a prezentační vrstva zobrazující obsah uživateli a umožňující mu ho tvořit či upravovat (frontend). Zde se dají uvést tři základní typy podle zdrojů [33, 37] (vizuální vyobrazení představovaných architektur je k vidění na obrázku 3.3):

- **Coupled** – Architektura, někdy také zvaná tradiční, spojující backend dohromady s frontendem. Redakční systémy stavěné tímto způsobem mají pevně propojenou správu obsahu a jeho prezentaci. Obsah je backendem publikován do přímo určeného frontendu, díky čemuž jsou kladeny menší nároky na kompletní zprovoznění systému. Nevýhodou takového přístupu je mimo jiné nemožnost snadné distribuce obsahu do kanálů, jež nepodporují zobrazení daného frontendu (chytré hodinky například běžně pohodlně nezobrazí HTML stránku).
- **Decoupled** – Decoupled architektura odděluje backend od frontendu. Frontendová část je dodávána spolu s backendem, ale její využití není vyžadováno. Backend je schopen komunikace pomocí aplikačního rozhraní (obecně popsáno dříve v úvodu sekce 3.3), což dovoluje vyvíjet vlastní frontendové aplikace, jež toto rozhraní využívají. To poskytuje flexibilitu ve výběru frontendových technologií a tím pádem i zařízení, na něž je možné obsah distribuovat.
- **Headless** – Redakční systémy s headless architekturou by se daly vnímat jako podmnožina decoupled systémů. Bývají ale poskytovány striktně bez frontendové části. Jedná se tedy pouze o backend ke komunikaci s databází a operaci s daty. Design, implementace a připojení frontendu je ponecháno čistě na vývojářích integrující CMS.



Obrázek 3.3: Diagramy základní podoby představených architektur redakčních systémů. Barevně jsou vyznačeny továrně dodávané součásti. Vyobrazený coupled systém má backend napřímo spojen s předem daným frontendem ve formě webové stránky. U decoupled systému je v tomto případě využíváno dodaného frontendu pro zobrazování obsahu na počítači, ale vlastní mobilní aplikace s backendem komunikuje pomocí dostupného aplikačního rozhraní. Headless systém na obrázku nabízí už pouze API. Jak frontendová aplikace pro počítače, tak pro mobily, jsou vlastní.

3.6 Současné redakční systémy

Redakční systémy lze podle zdroje [2] rozdělit do čtyř základních kategorií lišícími se způsobem distribuce a licencování:

- **Individuální systémy** – Tento typ řešení je specificky navržen a vytvořen podle konkrétních potřeb zadavatele. Výhodou je plná kontrola nad funkcionalitou a dalšími aspekty systému už od jeho koncepce. Nevýhodou může být typicky vyšší finanční a časová náročnost na vývoj a údržbu.

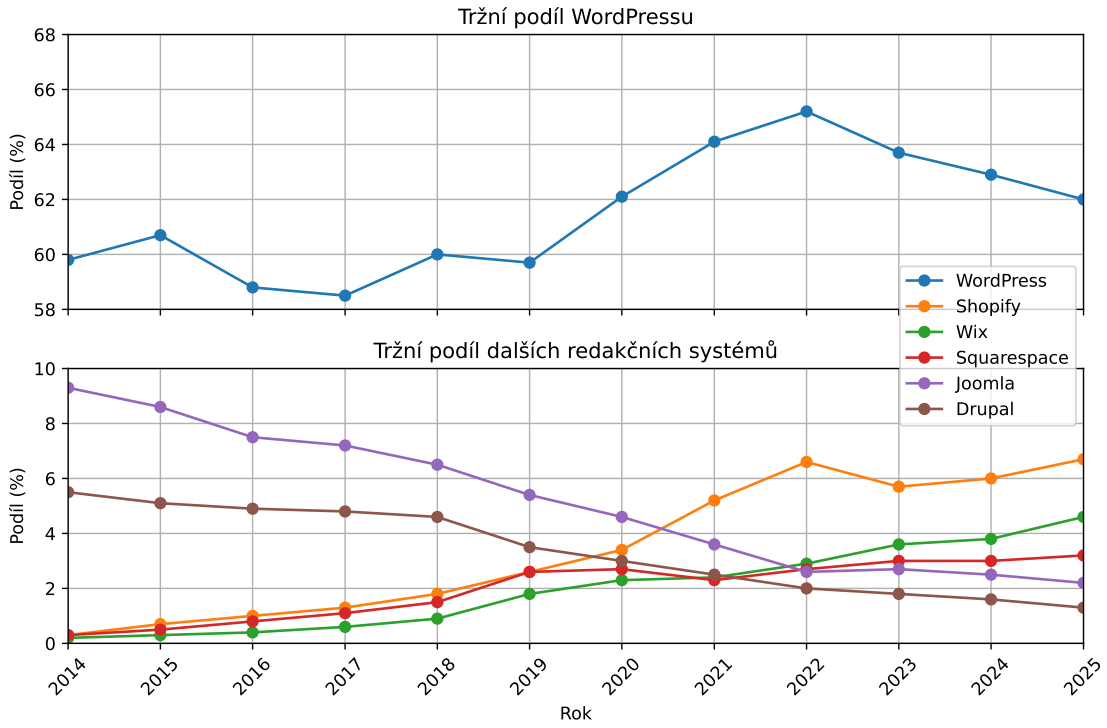
- **Komerční systémy** – Řešení nabízená veřejnosti za poplatek. Obvykle zahrnují individuální technickou podporu a servis od profesionálního týmu. Vývojáři těchto komerčních řešení můžou mít silnější motivaci stavět systémy s větším ohledem na pohodlnost užívání pro koncové uživatele, což by se ideálně mělo pozitivně promítat do počtu prodaných licencí.

- **Open-source systémy** – Tato řešení bývají nabízena obvykle zdarma, s otevřeným zdrojovým kódem a s velkou komunitní podporou, což jde ruku v ruce s často veřejně zdokumentovanými přístupy k nejčastějším problémům. Jejich nasazení a používání ale nemusí být vždy tak uživatelsky přívětivé jako v případě typických komerčních systémů.

- **Software jako služba** – Tento typ je též zvaný *SaaS* (z anglického *Software-as-a-Service*). Bývá poskytován jako online služba na bázi předplatného, která po uživatelích nevyžaduje investici do vlastní infrastruktury – systém je nasazen na serverech provozovatele, který se stará také o nové funkce a bezpečnostní aktualizace, a uživatelé k němu obvykle přistupují přes webový prohlížeč. Výhodou je například jednoduchost pořízení a nízké počáteční náklady. Nevýhodou může být závislost na provozovateli a omezená možnost přizpůsobení konkrétním potřebám. Jak uvádí zmíněný zdroj, hranice mezi tímto typem a dříve zmíněnými bývá často velmi tenká, jelikož někteří provozovatelé poskytují redakční systémy předchozích typů v režimu SaaS na vlastních serverech.

Tato sekce obsahuje přehled v současné době nejpoužívanějších redakčních systémů v oblasti webových aplikací a stránek. Důraz je kladen především na jejich administrační prostředí, která byla ve všech případech osobně vyzkoušena. Uvedeny jsou i nejdůležitější základní informace včetně distribučního modelu a zamýšleného zaměření. Mezi nejvíce zastoupené redakční systémy ve zmíněné oblasti podle [53] patří k lednu 2025 *WordPress*. Na obrázku 3.4 lze spatřit, že se svým 62% tržním podílem dalece převyšuje ostatní systémy, které se jinak pohybují pod 10 %. Jeho trend se navíc za posledních pět let zdá být poměrně stabilní, i když se momentálně začíná projevovat mírná klesající tendence. Další dva systémy – *Shopify* a *Wix* – během posledních let zaznamenávají výrazný růst v používání. Se svým 6,7%, respektive 4,6%, podílem na trhu CMS předběhly systémy *Joomla* a *Drupal*, které naopak svůj podíl viditelně ztrácí. Pokud by se ovšem bralo v potaz pouze tisíc nejnavštěvovanějších webů, Drupal by se umístil hned za WordPressem[11], tím pádem se stále jedná o oblíbený systém pro větší projekty a tato kapitola se mu věnuje detailněji. Dalším systémem, který postupem času získává na oblibě, byť pomalejším tempem, je *Squarespace*, jenž je s 3,2% podílem čtvrtým nejužívanějším a zmínku si zaslouží, i když v této sekci podrobněji rozebrán není.

[11] Vychází z porovnání individuálních tabulek každého jmenovaného systému v kategorii *Ranking* na webu https://w3techs.com/.



Obrázek 3.4: Srovnání podílu na trhu pěti nejpoužívanějších redakčních systémů. Kvůli vysokému rozdílu mezi podílem WordPressu a dalších systémů byl graf rozdělen na dva za účelem lepšího využití místa a přehlednějšího měřítka. Vodorovná osa je sdílena oběma grafy. Vertikální osa je individuální. Hodnoty převzaty z [53].

3.6.1 WordPress

WordPress je komunitně vyvíjený open-source redakční systém, jenž je v době psaní této práce jednoznačně nejpoužívanějším nástrojem pro tvorbu webových stránek a aplikací jak mezi samotnými redakčními systémy, tak v rámci celého internetu. Je odhadováno, že téměř 43 % všech webů na internetu používá právě tuto platformu [54]. Informace v této sekci byly čerpány ze zdrojů [34, 44] a vlastního testování, pokud není uvedeno jinak.

Jedním z důvodů, které napomohly masovému rozšíření WordPressu, může být jeho vysoká uživatelská přívětivost dovolující poměrně komplexní správu vlastních stránek a následnou publikaci obsahu bez znalosti programování. Pokud se však uživatel rozhodne zasahovat do kódu, má nad ním plnou kontrolu. Za používání tohoto systému navíc není nutné platit žádné licenční poplatky, což je potažmo další z řady věcí, jež přispěly jeho celkové popularitě, jako i jeho široká flexibilita a možnost využití pro blogy, komerční aktivity, prezentace podniků a další.

WordPress běží na decoupled architektuře, jak byla definována v sekci 3.5. Poskytuje tedy nepovinný vestavěný frontend úzce propojený s backendem. Součástí frontendu je veřejný modul zobrazující stránky určené ke čtení koncovým konzumentem a administrační rozhraní popsané dále v této sekci (existují zdroje, například [34], které považují administrační rozhraní jako součást backendu). Vzhled i funkcionalita veřejného modulu jdou upravovat tzv. *tématy* (anglicky *themes*), a to jak ve vizuálním editoru, tak podrobněji přímo v kódu. Různá témata jsou dostupná zdarma i komerčně. Mnoho jich je už v základu

responzivních a vhodných k prohlížení také na mobilních zařízeních. Použití vestavěného frontendu je dobré řešení pro technicky méně zdatné uživatele nebo pro projekty, kde by nemělo finanční či logický smysl hledat vlastní východisko. Přeje-li si uživatel systému od základu vytvořit vlastní frontendovou aplikaci, je ve WordPressu vestavěna podpora aplikačního rozhraní REST, pomocí níž původní backend může komunikovat s vlastním frontendem, případně s jinou aplikací dle konkrétního případu užití.

Implicitní administrační modul se nazývá *WordPress Dashboard*, jeho podobu znázorňuje obrázek 3.5. Jedná se o přehledné prostředí pro centrální správu daného webu. Kromě operací související s manipulací blogových článků a jejich publikací dovoluje například základní správu uživatelů včetně tvorby a nastavování jejich rolí, správu komentářů pod články, správu navigačního menu a jeho kategorií a další možnosti [45]. I základní možnosti úprav blogových příspěvků jsou poměrně flexibilní. Každý příspěvek může disponovat vlastním rozvržením textových bloků i grafických prvků.

Při záloze nebo migraci dat může pomoci vestavěný nástroj k jejich exportu a importu ve formátu XML, jenž umí pracovat třeba s příspěvky, komentáři či kategoriemi a dalšími entitami souvisejícími s obsahem. V základu však Dashboard bez jakýchkoliv zásahů nenabízí nástroje vhodné pro administraci složitějších aplikací, než jsou blogy a podobně jednoduché stránky. Úplně chybí například přímá správa samotné používané databáze a její migrace a zálohování. Nemluvě o eventualitě používání vlastního databázového schématu nebo dokonce o využití více databází současně k logickému oddělení dat tam, kde to situace vyžaduje. Nepozměněný WordPress pracuje s jedinou databází s předem definovanými tabulkami a atributy.

Zmíněné a další nedostatky je možné řešit pluginy. Ty jsou taktéž důležitou součástí WordPressu. Jsou to komunitní nebo komerční programová rozšíření základních funkcionalit systému, která jdou dynamicky přidávat i odebírat. V oficiálním pluginovém repozitáři na stránkách WordPressu[12] se momentálně nachází přes 59 tisíc pluginů zdarma a velké množství běžných problémů je možné vyřešit přímým importováním a potenciálně i úpravou některého z nich. Pluginy a jejich masové rozšíření jsou tedy dalším z prostředků, jimiž WordPress míří na uživatele bez znalosti programování, potažmo pokročilejším uživatelům jejich prostřednictvím může pomoci šetřit práci.

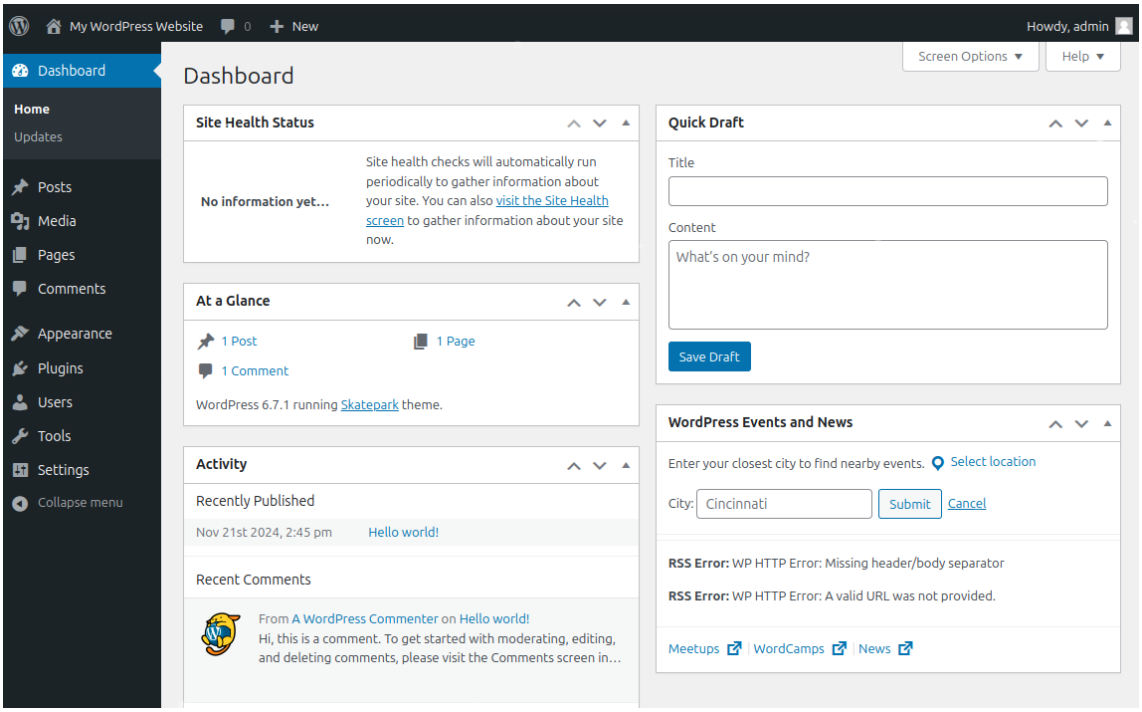
3.6.2 Shopify

Shopify je komerční redakční systém zaměřený především na tvorbu a správu elektronických obchodů těšící se vysoké oblibě zejména v posledních pěti letech [53]. Systém Shopify funguje na principu software jako služba. Výše pravidelného poplatku za užívání je stanovena úrovní zvoleného předplatného plánu[13]. Tato distribuční forma může vyhovovat hlavně uživatelům bez technických zkušeností, kteří díky ní nemusí řešit hosting, bezpečnostní aktualizace, výkon serveru nebo další záležitosti spojené se samotným provozem. Zápornou stránkou je již zmíněné finanční hledisko a oproti WordPressu také určitá ztráta kontroly nad určitými aspekty systému. Pokud není uvedeno jinak, následující text vychází z oficiální dokumentace [47] a vlastního vyzkoušení práce se systémem.

Architektura systému Shopify nejvíce připomíná decoupled architekturu, jež byla definována v sekci 3.5. Shopify disponuje vestavěným frontendem skládajícím se z administrátorské a veřejné části, k níž mají přístup zákazníci, tzv. storefront. Pokud je žádoucí plná kontrola nad vzhledem a funkcionalitou storefrontu, poskytuje Shopify aplikační rozhraní

[12] https://wordpress.org/plugins/

[13] Aktuální plány a jejich ceny jsou dostupné na adrese https://shopify.com/pricing

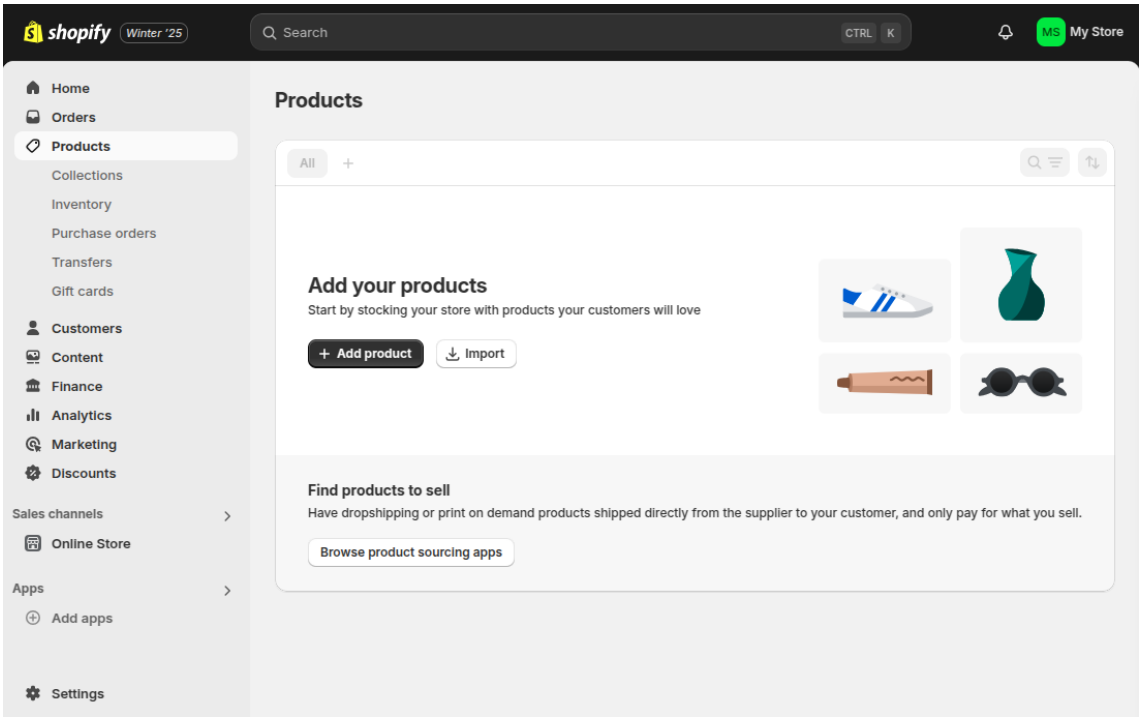


Obrázek 3.5: Snímek obrazovky administračního panelu systému WordPress. Tato tovární verze systému je přizpůsobena zejména pro blogování. V hlavním okně domovské stránky lze vidět panel pro rychlé vytvoření návrhu příspěvku. Postranní navigační menu pak poskytuje například záložky pro správu všech blogových příspěvků, mediálních souborů, jež lze do příspěvků vkládat, nebo uživatelských komentářů. Menu také dovoluje rychlý přístup k instalaci nových pluginů, díky nimž lze jednoduše přidat nové funkcionality přesahující původní zaměření na blogy.

na technologii GraphQL nazývané *Storefront API*, které je jedním z dvojice aplikačních rozhraní, jež Shopify nabízí (jeho protějšku bude věnován prostor dále v tomto textu), a jež lze využít pro komunikaci mezi backendem a vlastním storefrontem, která zahrnuje akce typické pro online nákupy jako načtení produktů nebo vytvoření objednávky (Storefront API tedy neslouží pro správcovské akce jako například úpravy popisků zboží). Pro tvorbu takovýchto vlastních aplikací Shopify poskytuje také framework postavený na webovém frameworku Remix s názvem *Shopify Hydrogen*[14] disponující komponentami optimalizovanými pro internetové obchody a komunikaci přes Storefront API. V případě, kdy postačí vestavěné řešení storefrontu, lze podobně jako u WordPressu vybírat z témat dostupných zdarma i za poplatek a ta případně následně upravovat ve vizuálním editoru nebo zásahem do jejich kódu [18].

Administrační rozhraní Shopify, znázorněno na obrázku 3.6, je stejně jako celý systém zaměřené především na obchodování, čemuž odpovídají také nástroje, jimiž disponuje. Patří mezi ně mimo jiné správa nabízených produktů, objednávek, refundací nebo různé druhy prodejních analýz. Nabízeno je i jednoduché rozhraní pro tvorbu blogových příspěvků. V administraci není možné řešit věci techničtějšího rázu jako například přímou správu databáze

[14] https://hydrogen.shopify.dev/



Obrázek 3.6: Snímek obrazovky administračního panelu systému Shopify. V postranním navigačním panelu lze vidět silné zaměření na komerci se záložkami pro správu objednávek, produktů, zákazníků a slev. Dostupná je však i záložka pro jednoduchou tvorbu blogových příspěvků. Sekce *Apps* slouží k instalaci a správě komunitních rozšíření, které se zbytkem systému z velké části sdílí orientaci na komerci.

nebo její zálohování. Je ale poskytována možnost importu a exportu dat o produktech, zákaznících, objednávkách a dalších byznysových objektech ve formátu CSV[15].

Samotné administrační rozhraní má uzavřený kód, nicméně stále umožňuje jistou míru úprav pomocí aplikačního rozhraní GraphQL nazývaného *Admin API*, které otevírá přístup k některým backendovým operacím a dovoluje vytvářet a dokonce vestavět do administračního rozhraní vlastní aplikace využívající Shopify backend ke zpříjemnění pracovního postupu. Tyto aplikace jdou rovněž sdílet na integrovaném tržišti. Sdílí tedy určitou podobnost se systémem pluginů ve WordPressu. V roce 2022 Shopify představilo také tzv. *Shopify Functions*. Díky nim je možné nahradit vybrané části backendové logiky vlastním kódem [38]. Byť tedy backend není zcela otevřený, pro pokročilejší uživatele je zde vyvíjena snaha alespoň o jeho částečné zpřístupnění.

3.6.3 Wix

Wix je druhým komerčním redakčním systémem, jehož obliba za posledních pět let výrazně vzrostla [53] zejména při vytváření menších projektů. Svým pojetím může připomínat dříve představený systém Shopify s rozdílem, že není stejně úzce zaměřen převážně na elektronické obchodování, ale snaží se poskytovat nástroje také pro tvorbu blogů, firemních prezentací, uměleckých portfolií a dalších typů webu. Wix taktéž využívá modelu software jako služba. Projekty jsou tím pádem hostovány na serverech provozovatele a uživatel platí měsíční nebo

[15] CSV je textový formát ukládající tabulková data.

roční poplatek o velikosti odvíjející se od zvoleného plánu. Navíc je poskytován permanentní bezplatný plán s určitými ústupky jako například menší úložný prostor nebo neodstranitelný reklamní banner. Pokud není v textu řečeno jinak, informace v této sekci pochází z oficiální dokumentace [56] a ze samostatného vyzkoušení systému.

Dalo by se argumentovat, že ze všech redakčních systémů představených v této sekci je Wix tím nejpřívětivějším pro uživatele bez jakýchkoliv technických znalostí. Disponuje umělou inteligencí, jíž uživatel může přirozeným jazykem sdělit, jaký typ webových stránek chce vytvořit včetně detailů o připravovaném projektu[16]. Umělá inteligence sama navrhne rozložení jednotlivých prvků, barevnou paletu nebo úvodní texty. Návrhy je možné dále ladit zadáváním upřesňujících požadavků. Pokud se uživatel rozhodne umělou inteligenci nepoužít, může si počáteční vzhled webu zvolit v rozsáhlém katalogu zdarma dostupných šablon pro různé typy webů či začít od nuly. Úpravy návrhu vytvořeného umělou inteligencí nebo vybrané šablony následně probíhají ve vizuálním editoru.

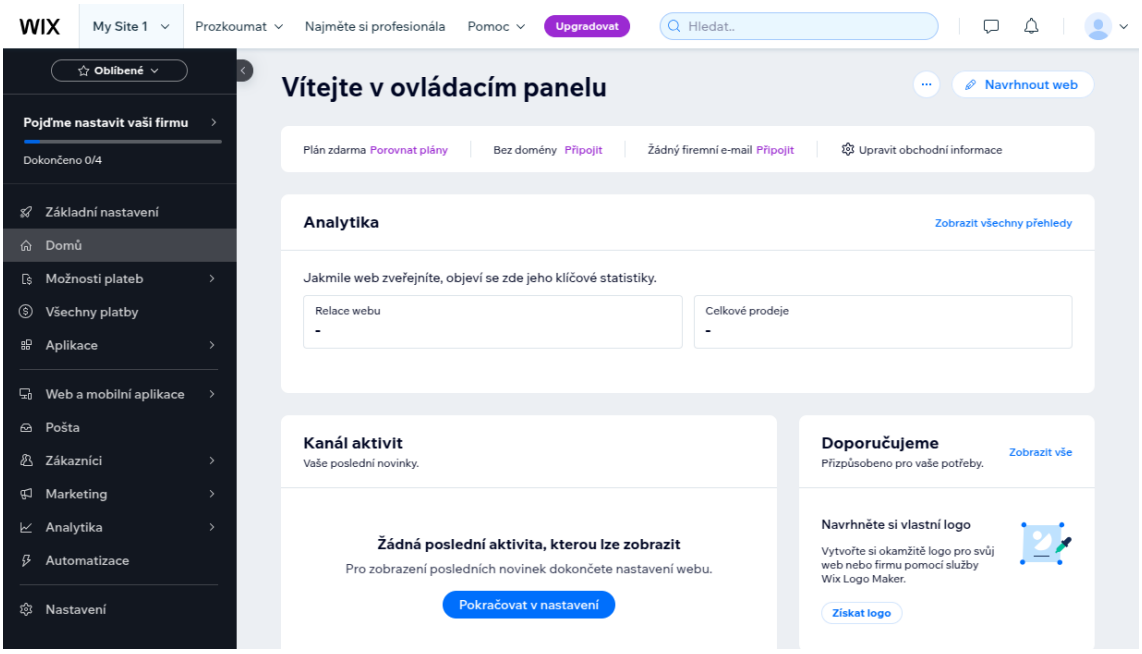
Pokročilejší uživatelé mohou vzhled a chování webu ovlivňovat také kódem. Pro dynamické prvky se dá využít oficiální vývojářskou sadu nazvanou *Velo*, jež dovoluje vkládat vlastní kusy kódu v programovacím jazyce JavaScript a ty spouštět jak v prohlížeči klientů tak na backendových serverech systému Wix. S určitými omezeními je povoleno používat i vlastní CSS kód[17]. Na rozdíl od Shopify je pomocí tohoto vývojářského nástroje umožněna také správa kolekcí v přidružené databázi MongoDB nebo připojení vlastních externích databází včetně tradičních relačních DB jako MySQL. Velo lze také uplatnit k vystavení aplikačního rozhraní REST, pomocí něhož s backendem webu mohou komunikovat venkovní aplikace. Pokud je účelem integrace vlastního frontendu s backendem Wixu, jeví se jako dobré řešení použít *Wix Headless*, tedy odlehčenou variantu služby ochuzenou o vestavěný frontend.

Součástí vestavěného frontendu je také administrační panel poskytující grafické rozhraní pro správcovské akce, který je viditelný na obrázku 3.7. Kromě vizuálního editoru webu je velké množství jeho základních funkcí zaměřeno na obchod a patří mezi ně například správa plateb, faktur, zákazníků a reklamních kampaní. Dále pak analýzy optimalizace pro internetové vyhledávače, návštěvnosti, chování uživatelů nebo úspěšnosti marketingu. Pro odemčení dalších možností se počítá s použitím *App Marketu*, kde lze nalézt rozšiřující aplikace, obdobu pluginů z WordPressu, začlenitelné jako nové záložky a funkcionality do administrace nebo jako dynamické prvky pro veřejnou část webu. Mezi aplikacemi jsou nabízeny namátkově podpora správy blogových příspěvků, integrace uživatelských komentářů nebo vystavování a prodej produktů[18]. Aplikace jsou dostupné zdarma i s cenovkou a vytvářeny jak oficiálním týmem Wixu, tak komunitou. Z hlediska počtu však zaostávají za ostatními porovnávanými systémy.

3.6.4 Drupal

Drupal je bez poplatků dostupný open-source redakční systém vyvíjený komunitou, jehož celkový podíl na trhu je od roku 2019 ve stálém úpadku. Zatímco momentálně nejpopulárnější systémy soustřeďují svoji pozornost na přívětivost k technicky méně znalým uživatelům, Drupal se snaží nabízet pokročilé nástroje, robustní zabezpečení a možnosti přizpůsobení [41], díky čemuž stále nachází své uplatnění převážně u zkušených jednotlivců či ve větších vývojářských týmech pracujících na náročnějších projektech vyžadujících vyšší

[16] Informace o této funkci jsou dostupné na https://wix.com/ai-website-builder.
[17] CSS je jazyk pro stylování elementů v HTML dokumentu.
[18] Všechny dostupné aplikace je možné prohlédnout na https://wix.com/app-market



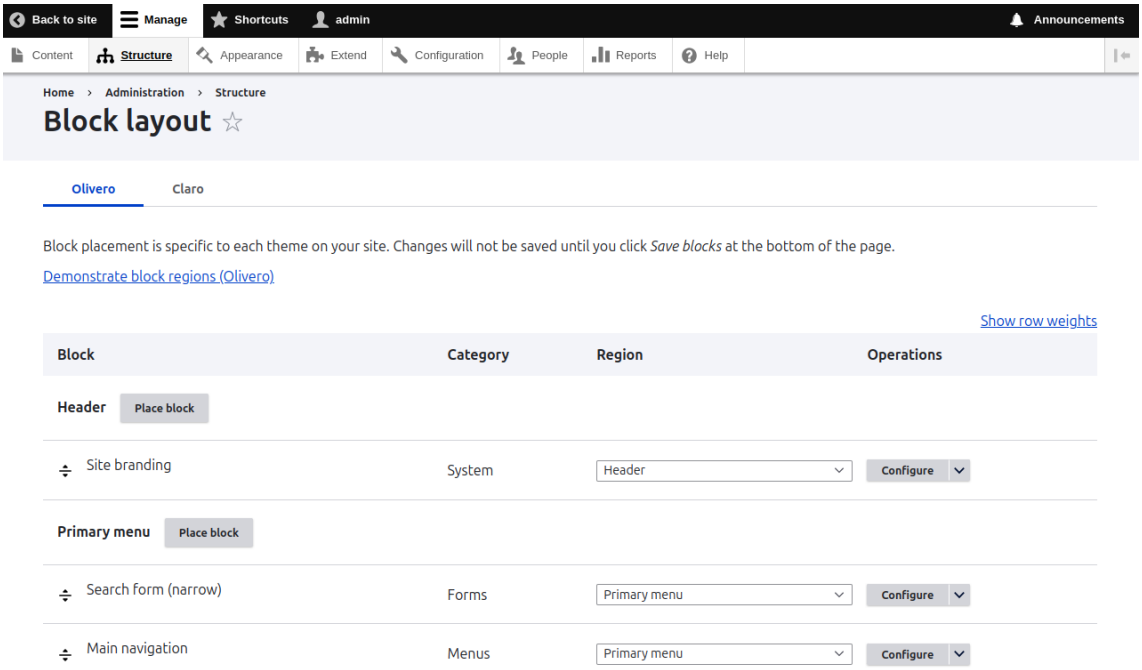
Obrázek 3.7: Snímek obrazovky administračního panelu systému Wix. Panel je bez základního přizpůsobení poměrně skoupý na funkcionalitu spojenou s jakoukoliv správou obsahu. Počáteční průvodce však může pomoci s výběrem vhodných rozšiřujících aplikací dle zaměření webu. K těmto aplikacím se dá dostat také přes online tržiště dostupným skrze proklik z postranního menu. Zajímavá je možnost vytvoření vlastní nativní mobilní aplikace přímo v administrátorském panelu. Pro její zveřejnění na standardních mobilních obchodech je však vyžadováno samostatné předplatné.

míru flexibility a škálovatelnosti, což ho v této oblasti i nadále činí druhým nejsilnějším hráčem za WordPressem. Vývojářská komunita okolo Drupalu ovšem nechtěla v podpoře běžných uživatelů zaostat za ostatními redakčními systémy a v lednu 2025 představila první verzi *Drupal CMS*, tedy předkonfigurované platformy, která má už v základu nabízet hotové řešení pro nevývojáře obsahující předinstalované nástroje k tvorbě webu bez potřeby znalosti programování. Původní varianta pro pokročilé uživatele je dále nabízena pod názvem *Drupal Core* [9].

Drupal Core

Drupal Core vyniká především svojí vysokou mírou přizpůsobitelnosti, která uživatelům umožňuje upravit prakticky každý aspekt webových stránek dle potřeby ať už přímými úpravami kódu, nastavením v poměrně komplexním administračním panelu nebo použitím povětšinou zdarma dostupných modulů [3], obdoby pluginů WordPressu, jež ovšem nenabízí pohodlnou integraci jedním kliknutím v administračním panelu, ale pouze instalaci přes software pro správu knihoven jazyka PHP *Composer*. Ačkoliv bylo zmíněno, že Drupal Core podporuje úpravy v grafickém rozhraní, nemusí nutně jít o na první pohled intuitivní záležitost. Systém ve svém administračním panelu umožňuje vytvářet vlastní typy obsahu detailní definicí jejich struktury například povinnými i nepovinnými textovými poli, obrázky, odkazy nebo vztahy mezi jednotlivými obsahovými typy, a to včetně toho, jakým způsobem se bude každý druh obsahu zobrazovat v jaké části stránky nebo jaké skupiny uži-

vatelů k němu budou mít přístup. Obsah je umisťován do uživatelem definovaných bloků. Skládáním těchto bloků potom vzniká rozložení stránek [24]. Drupal Core však bez dodatečné aktivace nedisponuje vizuálním editorem a rozložení je nastavováno v tabulkovém rozhraní (k vidění na obrázku 3.8). S procesem může do určité míry pomoci použití jednoho z převážně volně dostupných komunitních témat, která poskytují předdefinovaná rozložení. Kromě zmíněné správy obsahu a struktury webu obsahuje administrační rozhraní nástroje pro správu uživatelů a prohlížeč zaznamenaných oznámení, chyb a logů z různých komponent systému. V nezměněné instalaci grafické rozhraní neumožňuje v žádné míře provádět správu ani zálohu používaných databází [3]. Drupal Core má zastoupení i v decoupled aplikacích, kde je používán pouze jeho backend ve spojení s vlastním frontendem [50].



Obrázek 3.8: Snímek obrazovky administračního panelu systému Drupal Core. Na fotce lze vidět způsob, jakým se v Drupal Core nastavuje rozložení jednotlivých prvků na stránce. Grafický editor není v továrním nastavení systému aktivovaný. Aktivaci lze provést ze záložky *Extend*, spolu se spuštěním některých dalších modulů dodávaných zároveň s instalací systému. Na rozdíl od předchozích představených systémů však Drupal Core nedisponuje vestavěným tržištěm a většinu komunitních modulů je třeba stahovat a instalovat manuálně.

Drupal CMS

Nová varianta Drupal CMS je ke dni 16. 1. 2025 dostupná méně než jeden měsíc a neexistuje velké množství o ní pojednávající literatury, proto je zdrojem všech následujících informací oficiální uživatelská příručka [19] a osobní testování. Tato varianta přímo staví na základu Drupal Core a nepřichází o původní možnosti přizpůsobitelnosti. Zjednodušuje původní nastavení tím, že se uživatele zeptá, o jaký typ webové aplikace má zájem, a podle výběru předkonfiguruje systém včetně instalace předpokládaně užitečných modulů. Dalším velkým krokem směrem k běžnému uživateli je intuitivněji působící rozhraní, v němž je na rozdíl od Drupal Core možné upravovat rozložení a obsah jednotlivých stránek ve vizuálním

editoru. Do rozhraní byla přidána také možnost instalování modulů jedním kliknutím po vzoru WordPressu, byť ne všechny moduly se v této době dají integrovat tímto způsobem[19]. Představeny byly také *recepty* (anglicky *recipes*) – balíky modulů, konfigurací a přednastavených typů obsahu, jež lze taktéž kliknutím přímo v grafickém rozhraní zakomponovat do systému. Drupal CMS dává k dispozici například recept na webové portfolio, který po instalaci zpřístupní umělecké projekty jako typ obsahu, konfiguraci k zobrazování médií ve vysokém rozlišení a nastavení nástroje pro úpravu rozložení stránek tak, aby podporoval zobrazování projektů v mřížce.

3.6.5 Shrnutí

Představeny byly čtyři populární redakční systémy s různým zaměřením, distribučním modelem i úrovní požadované technické zdatnosti. Porovnání základních vlastností zajišťuje tabulka 3.1. WordPress a Drupal reprezentují open-source řešení s možností plné kontroly nad systémem a daty. Jedná se o flexibilní nástroje pro publikaci obsahu, které v rukou zkušeného uživatele nabízí vysokou mírou přizpůsobitelnosti a mohou být vhodné i pro náročnější projekty. Shopify je systém typu software jako služba zaměřený především na elektronickou komerci, čemuž přizpůsobuje i své nástroje, jež jsou bohaté na prodejní analýzy či správu zboží a zákazníků. Wix je rovněž systém typu software jako služba. Nejvhodnějším použitím se jeví být tvoření jednodušších internetových prezentací, portfolií nebo eshopů. Díky vestavěným asistentům používajícím umělou inteligenci dovoluje sestavit web i uživatelům bez hlubšího technického zázemí.

	WordPress	**Shopify**	**Wix**	**Drupal Core**
Typ distribuce	Open-source	Software jako služba	Software jako služba	Open-source
Zaměření	Univerzální	Online obchody	Menší weby, prezentace, portfolia	Rozsáhlé weby
Obtížnost	Střední (záleží na pluginech)	Nízká	Nízká	Vysoká
Modifikovatelnost	Vysoká	Částečná	Částečná	Vysoká
Rozšiřující moduly (počet)	Ano (59 000)	Ano (12 000)	Ano (800)	Ano (53 000)
Cena	Zdarma	Předplatné	Zdarma / Předplatné	Zdarma

Tabulka 3.1: Stručné porovnání některých základních vlastností představených redakčních systémů. V tabulce byla zanedbána varianta Drupal CMS, která je ke dni 16. 1. 2025 k dispozici po dobu kratší než jeden měsíc a některé vlastnosti by bylo složité v takto krátkém horizontu přesně určit.

[19] V době psaní této práce je v Drupal CMS zhruba 3000 z celkových 53000 modulů dostupných k instalaci jedním kliknutím.

Kapitola 4

Současný stav a analýza požadavků

Projekt Zastupko.cz se zrodil jako součást diplomové práce [57] a původně sloužil pro analýzy a přehledné vizualizace dat z hlasování Zastupitelstva města Brna. Během navazující diplomové práce [30] bylo do projektu uvedeno zobecnění pro další municipality spolu s grafickým administrátorským rozhraním dovolujícím jejich správu i správu podřízených politických subjektů, členů a záznamů ze zasedání jejich zastupitelstev, které byly do té doby spravovány pomocí skriptů a přímých zásahů do databázového serveru. Při představování současného stavu bude důraz kladen na tento administrátorský modul, jelikož jeho rozšíření je hlavním praktickým výstupem této práce. Požadavky na toto rozšíření jsou uvedeny v závěru kapitoly v sekci 4.6. Následující odstavce čerpají z uvedených diplomových prací, z vlastní analýzy zdrojových kódů, uživatelského rozhraní aplikace a konzultací s týmem projektu Zastupko.cz.

4.1 Představení projektu a cílových uživatelů

Projekt Zastupko.cz si dává za úkol vzbudit v uživatelích zájem o politiku, zejména komunální a krajskou, a přispívat k transparentnosti státní správy. Původní práce [57] udává jako cílové uživatele muže i ženy jakékoliv profese ve věku od 18 let mající volební právo. Jde o širokou skupinu s předpokladem výrazných rozdílů v technických znalostech, proto je při jeho vývoji kladen význam na smysluplnou a co možná nejpřístupnější vizuální prezentaci.

Aplikace umí pro kraje a vybraná města vizualizovat data ze zasedání zastupitelstev a dalších politických orgánů za aktuální i předcházející volební období (za podmínky, že jsou pro municipalitu potřebná data dostupná a zadaná v systému). Systém pro každé zasedání dovoluje zobrazit přítomnost zastupitelů, výsledky jednotlivých hlasování a to, jak hlasovali dané politické subjekty i konkrétní jednotlivci. Mimo to si uživatel může zobrazit analýzy, jež zahrnují výpočty docházky subjektů a členů za celé volební období. Dále je možné analyzovat shodu hlasů mezi vybranými členy patřícími do stejných i různých subjektů či nejednotnost v hlasování subjektů, tedy v jaké míře členové jedné strany vybírají rozdílné hlasovací možnosti. V neposlední řadě si lze v analýzách zobrazit kohezi jednotlivých hlasování, tedy to, jaká u hlasování panovala shoda mezi všemi přítomnými hlasujícími napříč subjekty.

Pro zadávání dat do systému slouží administrační modul, který není zamýšlen pro širokou veřejnost. Úpravy pomocí tohoto modulu jsou povoleny provádět pouze uživatelům majícím přístup k účtu s právy globálního administrátora a členům disponujícím běžným účtem, jimž bylo přiděleno právo upravovat jednu či více municipalit. Modul byl doposud

využíván pouze členy týmu projektu. Z důvodu udržitelnosti projektu by ale bylo lepším řešením svěřit správu některých municipalit třetím stranám, čímž by se dostal do rukou potenciálně technicky méně zdatných uživatelů, na což je třeba při pokračujícím vývoji dbát zřetel. Schopnosti administračního modulu a jeho uživatelských rolí jsou detailněji popsány v následující sekci.

4.2 Funkce administračního modulu

Administrační modul je zpřístupněn po přihlášení uživatelským účtem, jehož vytvoření je zdarma dostupné komukoliv. Běžnému registrovanému uživateli je povoleno za určitých podmínek popsaných níže upravovat data zobrazovaná ve veřejných vizualizacích. Nadřazeným typem účtu je globální administrátor. Tato role má v systému nejvyšší práva, je přidělována pouze účtům vybraných členů týmu projektu a její specifika jsou rovněž přiblíženy níže. Neregistrovaný uživatel do administračního systému nemá přístup a je omezen na pouhé čtení veřejného modulu, kterým se tato sekce nezabývá. Jeho roli je proto v následujícím popisu možné zanedbat.

4.2.1 Registrovaný uživatel

Běžný typ účtu lze získat po volně dostupné registraci. Nově vytvořený účet nemá možnost jakkoliv zasahovat do dat, která se promítají do veřejného modulu. Takový účet se může pokusit získat práva odesláním žádosti o správu existující municipality nebo vyplněním formuláře k založení nové a jejím odesláním administrátorům.

Pokud je žádost přijata globálním administrátorem, běžný účet se stává správcem existující, respektive nově přidané municipality a může nad touto municipalitou provádět všechny úpravy. Mezi ty patří libovolná editace atributů municipality jako její název, popis, logo a další. Zpřístupněny jsou také možnosti úprav přidruženého zastupitelstva jako správa politických subjektů a členů zastupitelstva, tedy jejich vytváření, úpravy a mazání. Tyto entity nejsou chráněny proti zlovolnému smazání, jelikož tato akce přímo odstraní záznam v databázi.

Účet s povolením spravovat municipalitu dále může v jejím zastupitelstvu vytvářet funkční období a přidávat členy aktivní při startu tohoto období. Pod jednotlivá období je možné následně vkládat záznamy ze zasedání včetně upřesnění členů, jejichž členství bylo aktivní v době daného zasedání (toto je využito k přiřazení příslušnosti členů k politickým subjektům, které je prováděno automaticky). K zasedáním lze vytvářet záznamy proběhnutých hlasování, u nichž se kromě dalších informací zadává vybraná hlasovací možnost každého přítomného člena a celkový výsledek. Hlasování lze také vymazat, přičemž je opět provedeno kompletním smazáním z databáze.

4.2.2 Globální administrátor

Povýšit běžného registrovaného uživatele na globálního administrátora lze pouze změněním příslušného příznaku přímo v databázi. Tento typ účtu je vyhrazen pouze pro důvěryhodné členy týmu projektu, jelikož má přístup prakticky ke všem možnostem, které systém nabízí. Globálním administrátorům je povoleno vytvářet nové municipality a zároveň s nimi jejich zastupitelstva dle libosti. Dále mohou provádět veškeré editace dat, které jsou povoleny i registrovaným uživatelům, s rozdílem, že nepotřebují nejprve získat povolení ke správě dané municipality. Automaticky mohou editovat všechny municipality a zastupitelstva v systému.

Mazání municipalit je také povoleno, při této akci však nedochází k reálnému smazání z databáze, pouze zneviditelnění v samotné aplikaci.

Kromě výše uvedeného je administrátorům dovoleno nahlížet na žádosti podané běžnými uživateli a rozhodovat o jejich schválení. Jak již bylo řečeno v předchozí podsekci, existují dva typy žádostí – žádost o vytvoření nové municipality a žádost o přidělení práv ke správě existující municipality.

4.3 Systémová architektura a použité technologie

Projekt je implementován jako webová aplikace na základě architektury klient-server (přiblížena výše v sekci 3.2). Vizuálně architekturu představuje obrázek 4.1. Je složena z těchto základních komponent:

- **Server**
  Backendová součást aplikace postavená na frameworku Flask[1] programovacího jazyka Python. Přijímá a odpovídá na požadavky frontendové části pomocí aplikačního rozhraní na principu REST (princip popsán v sekci 3.3). Čte a upravuje stav databází. Je rozdělen na moduly poskytující rozhraní pro veřejné vizualizace a pro administrátorské operace.

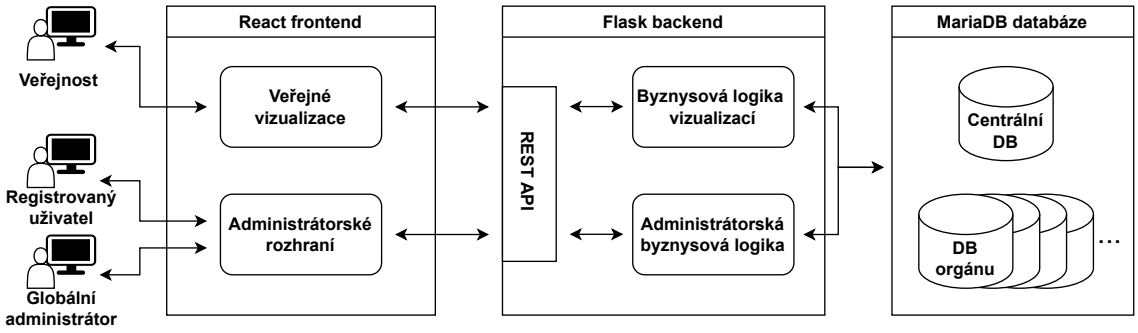
- **Klient**
  Frontendová součást aplikace je samostatná a od backendu programově oddělená komponenta. Využívá knihovny React[2] pro programovací jazyk JavaScript. Vykresluje grafické rozhraní a na základě příkazů od uživatele odesílá požadavky na server. Podobně jako backend disponuje oddělenými moduly pro uživatelské rozhraní veřejné a administrátorské části.

- **Databáze**
  Relační databáze MariaDB[3]. Ukládá data o municipalitách, zastupitelstvích, hlasováních a dalších entitách potřebných k chodu aplikace. Používány jsou dva typy databází lišící se svými schématy – centrální schéma pro metadata o municipalitách a schéma pro databáze orgánů, kterých systém používá vícero, pro data samosprávných orgánů (datový model je více přiblížen v sekci 4.4).

Adresářová struktura

Systém je rozdělen na dvě samostatné aplikace – backendovou a frontendovou – pro něž jsou v kořenovém adresáři projektu vyhrazeny zvláštní složky `api/`[4], respektive `react-frontend/`. Ke splnění praktických cílů této práce bude třeba zasahovat do obou součástí, proto tato sekce krátce představuje jejich základní strukturu. Vykreslení adresářové struktury frontendu bylo převzato z diplomové práce [30] a upraveno.

[1] https://flask.palletsprojects.com/en/stable/
[2] https://react.dev/
[3] https://mariadb.com/
[4] Název je pozůstatkem z počátků projektu.



Obrázek 4.1: Zjednodušené schéma architektury systému. Frontend i backend jsou rozděleny na veřejnou a administrátorskou část, které pracují více či méně samostatně. Používány jsou dva typy databází – centrální a orgánová. Schéma bylo načrtnuto ve spolupráci s autorem práce [29].

Backend

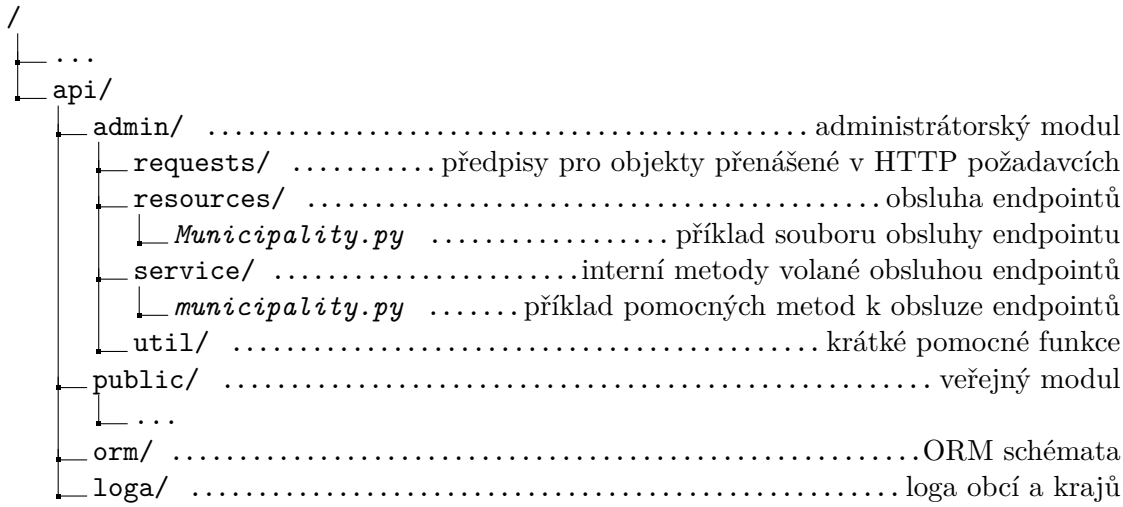
Backendová část momentálně obsahuje dva programové moduly – administrační a veřejný ve složkách `admin/` a `public/`. Druhý jmenovaný se stará o páteřní procesy spojené s načítáním vizualizací v části aplikace určené pro širokou veřejnost. Účelem této práce není zasahovat do veřejného modulu, proto se jím následující text nezabývá.

Adresářová struktura administrátorského modulu je k vidění na obrázku 4.2. Modul je tvořen čtveřicí adresářů vnořených do složky `admin/` obsahující zdrojové kódy programovacího jazyku Python. Většina logiky se nachází v adresářích `resources/` a `service/`, z nichž první uschovává řadu souborů, jejichž smyslem je obsluhovat endpointy aplikačního rozhraní, respektive obsahují třídy, jejichž metody jsou přímo volány při požadavku na daný koncový bod. Druhá jmenovaná složka obsahuje třídy definující interní metody určené k volání při obsluhování endpointů. Většinově jde o na více místech používané metody přímo interagující s databází. Posledním pro modul zásadním adresářem je o úroveň výše se nacházející `orm/`, jenž slouží rovněž veřejnému modulu a ukládá schémata objektově relačního mapování pro přirozenější práci s databázovými objekty uvnitř kódu.

Frontend

Adresářová struktura frontendu je znázorněna obrázkem 4.3. Na první pohled neposkytuje tak jasné rozdělení na veřejný a administrátorský modul. Ačkoliv však soubory veřejné části nejsou viditelně odděleny, prakticky celý administrátorský modul se nachází v adresáři `/react-frontend/src/admin/`, kde jsou v souboru `init.js` stranou definované také jeho vlastní směrovací URL adresy k zobrazování stránek. To umožňuje provádět jeho úpravy bez jediné změny zasahující do hlavní části aplikace, která vykresluje vizualizace běžným uživatelům. Výstupem práce nebudou úpravy této veřejné části, proto její adresáře nebudou v následujícím textu zmiňovány.

Každá stránka, kterou má administrátor či registrovaný správce možnost vidět, má vlastní soubor s kódem frontendové knihovny React pro programovací jazyk JavaScript, který určuje, co a za jakých podmínek bude uživateli vykreslováno. Tyto soubory jsou do adresářů strukturovány způsobem, jenž člení související stránky do shrnujících složek. To napomáhá orientaci tím, že pro příklad stránky zobrazující seznam municipalit, úpravu municipality a přidání municipality jsou seskupeny ve stejném adresáři. Každá skupina má



Obrázek 4.2: Adresářová struktura serverové části aplikace napsané ve frameworku Flask. Pro text nedůležité složky a soubory byly zanedbány.

v tomto adresáři současně k dispozici vlastní specifické komponenty grafického rozhraní (formuláře, tlačítka apod.) ve složce `components/`. Globální komponenty využívané vícero stránkami administračního rozhraní jsou ve stejnojmenné složce o úroveň výše.

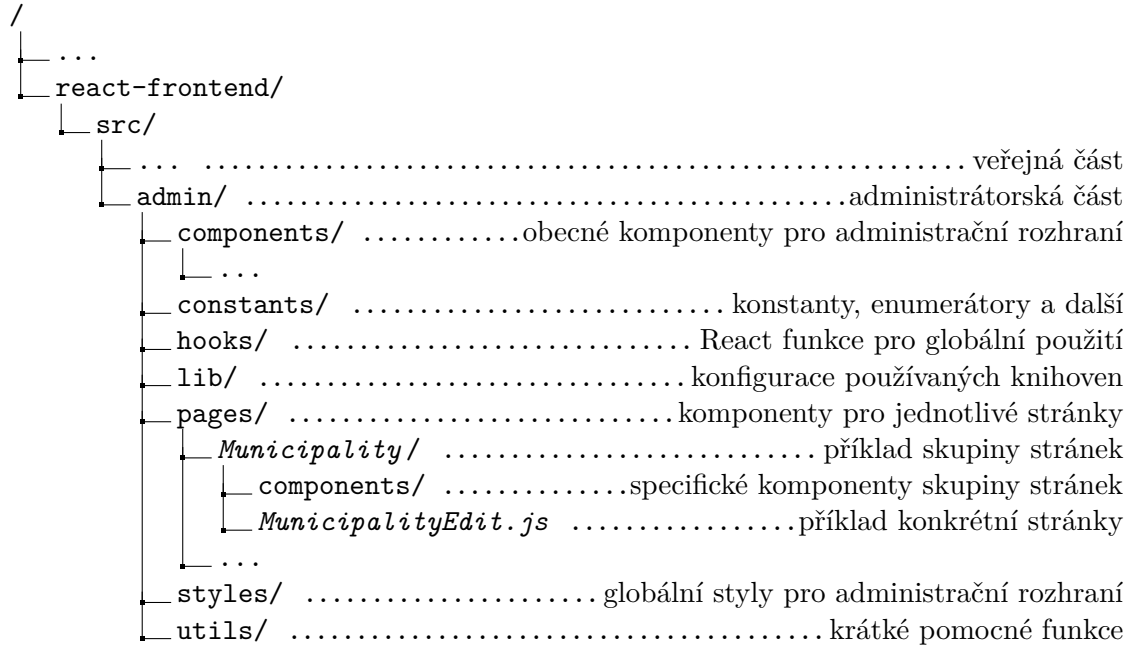
4.4 Datový model

Během zobecňování systému a zavádění podpory pro další municipality mimo město Brno v rámci práce [30] muselo dojít k poměrně výrazným změnám na úrovni práce s databází. Ačkoliv byl původní datový model navrhnut s dostatečnou obecností pro podporu ukládání hlasování různých municipalit [57], nepočítal s možností pokrytí více municipalit v jedné instanci aplikace. Používal proto jedinou databázi k ukládání záznamů hlasování a dalších potřebných dat právě jedné municipality, respektive jednoho jejího samosprávného orgánu.

Navazující práce problém vyřešila přidáním centrální databáze uchovávající mimo jiné metadata o jednotlivých pokrývaných municipalitách a databáze orgánu, která je v práci [30] nazývána databází municipality. Tato databáze užívá rozšířené obdoby původního databázového schématu a v systému se vyskytuje ve více instancích – jedna pro každý orgán spadající pod sledované municipality. K jedné municipalitě může patřit více takovýchto databází, pokud v systému disponují více orgány, proto může být název databáze municipality zavádějící a tento text se bude držet nového označení.

4.4.1 Centrální databáze

Centrální databáze v systému slouží pro ukládání metadat o municipalitách, orgánech, uživatelích a jejich právech spravovat municipality. Jednotlivé databázové entity jsou uvedeny níže a jejich vztahy jsou zobrazeny na obrázku 4.4. Názvy entit byly oproti reálně používaným pro lepší čitelnost lehce upraveny přidáním diakritiky a použitím kapitalizace místo podtržítek:



Obrázek 4.3: Adresářová struktura klientské části aplikace napsané s pomocí knihovny React. Pro text nedůležité složky a soubory byly zanedbány.

- **Municipalita**
  Obsahuje informace o municipalitě, mimo jiné její textový identifikátor, název, kontaktní emailovou adresu nebo odkaz na webové stránky. Její typ je určen odkazem na enumerátor **TypMunicipality**. Některé z možností jsou *obec*, *město*, *kraj* a další.

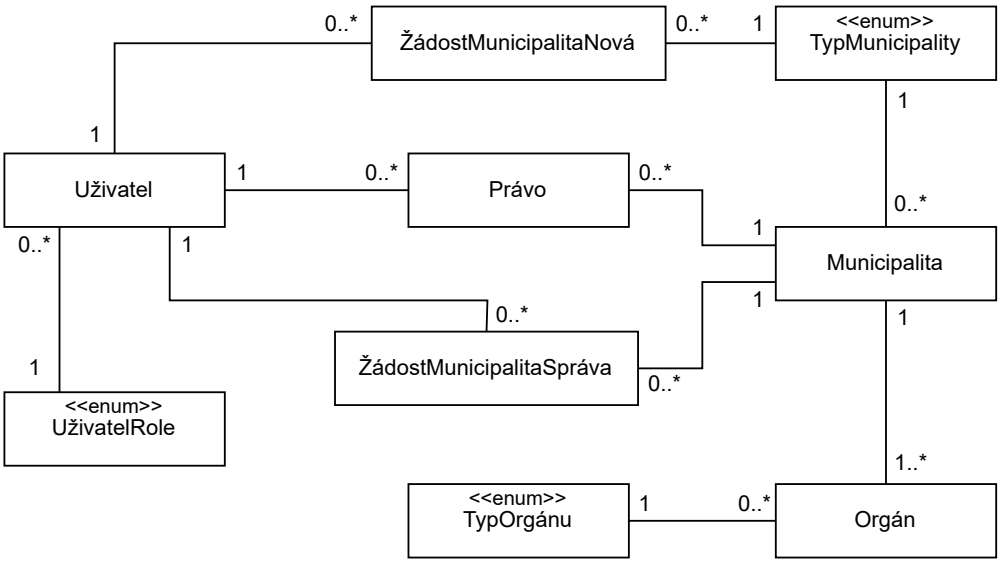
- **Orgán**
  Je vázán na konkrétní municipalitu. V databázi se ukládají informace jako název nebo počet zastupitelů. Typ je určován enumerátorem **TypOrgánu**. V tento moment jsou jediné dostupné typy *zastupitelstvo* a *rada*.

- **Uživatel**
  Záznam registrovaného uživatele. Ukládáno je mimo jiné jeho přihlašovací jméno a zašifrované heslo. Typ uživatele se určuje enumerátorem **UživatelRole**. Momentálně jsou rozeznávány role *user* a *admin*, z nichž první reprezentuje běžného uživatele a druhá administrátora.

- **Právo**
  Určuje právo běžného uživatele spravovat municipalitu a její orgány. Uživatelé s rolí administrátora mají implicitní právo spravovat všechny municipality.

- **ŽádostMunicipalitaNová**
  Reprezentuje žádost běžného uživatele o vytvoření a práva ke správě nové municipality. Nese v sobě informace o požadované municipalitě a jejím prvním orgánu.

- **ŽádostMunicipalitaSpráva**
  Reprezentuje požadavek běžného uživatele o získání správcovského práva k existující municipalitě.



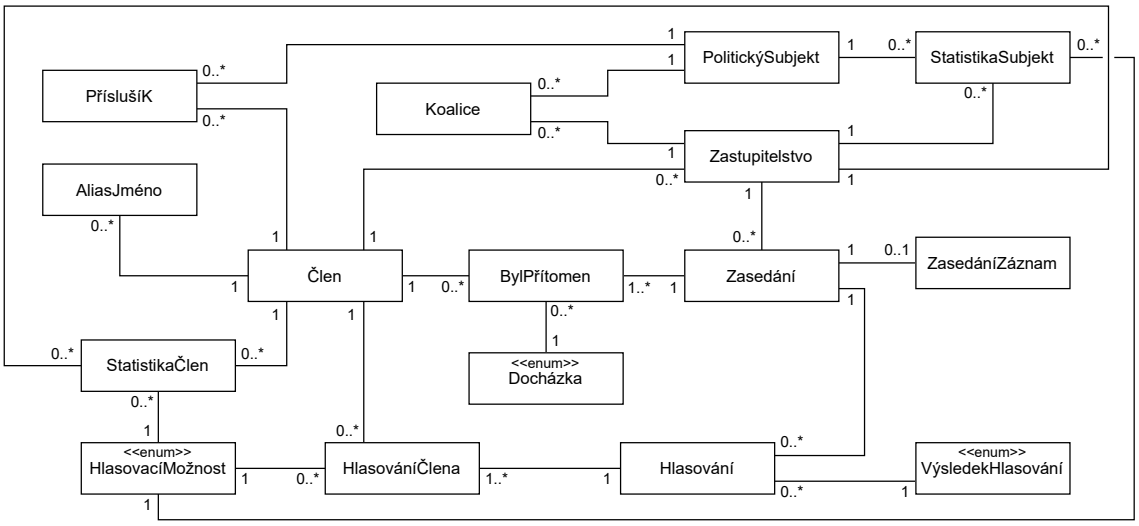
Obrázek 4.4: Schéma entit centrální databáze. Atributy byly vynechány k udržení kompaktnosti. Názvy byly pro lepší čitelnost mírně upraveny. Entity *«enum»* značí tabulky s výčty možností.

4.4.2 Databáze orgánu

Účelem databáze orgánu je ukládat informace o politických subjektech, členech, zasedáních a hlasováních uvnitř jednoho samosprávného orgánu. Jak bylo řečeno v úvodu této sekce, takovýchto databází se v systému nachází vícero, jelikož každý orgán pod sledovanými municipalitami disponuje jednou takovou. Databáze orgánů musí být pojmenovány podle vzorce *<orgán>_<municipalita>*, kde *<orgán>* představuje zástupný symbol pro typ orgánu (musí se nacházet v enumerátoru **TypOrgánu** v centrální DB) a *<municipalita>* pro textový identifikátor municipality. Příkladem validního jména databáze orgánu je `zastupitelstvo_brno` či `rada_most`. Níže následuje výpis databázových entit z databáze orgánu, jejichž vztahy lze vidět na obrázku 4.5. Názvy entit byly opět mírně upraveny pro lepší čitelnost přidáním diakritiky a použitím kapitalizace místo podtržítek. Vynechány byly entity **Alias**, **HlasováníObsahujeSlovo**, **KlíčovéSlovo**, **Oblast**, **PředvolebníSlib**, **Slíbil**, **SlibObsahujeSlovo** a **Zdroj**, neboť ačkoliv se v databázovém schématu nachází v současné chvíli nejsou používány a neexistuje pro ně podpora v žádné části aplikace.

- **PolitickýSubjekt**
  Uchovává název, zkratku a kód barvy zvolené k reprezentaci daného subjektu, tedy politické strany nebo koalice stran. K zaznamenávání statistik o počtu užití jednotlivých hlasovacích možností všemi členy subjektu je entita **StatistikaSubjekt**.

- **Člen**
  Ukládá jméno, příjmení a pohlaví člena orgánu. Jmenné aliasy členů mohou být definovány entitou **AliasJméno**. Entita **StatistikaČlen** slouží k počítání celkového součtu užití jednotlivých hlasovacích možností členem. Příslušnost k politickému subjektu může být vyjádřena pomocí **PříslušíK**.



Obrázek 4.5: Schéma entit databáze orgánu. Atributy a nepoužívané entity byly vynechány k udržení kompaktnosti. Názvy byly pro lepší čitelnost mírně upraveny. Entity *«enum»* značí tabulky s výčty možností.

- **Zastupitelstvo**
  Reprezentuje jedno funkční období. Aktuální název je pozůstatek z původní verze systému, kde byl dostatečně výstižný. Uchovává mimo jiné datum začátku a konce funkčního období a informaci o leaderovi orgánu v tomto období. Koaliční strany jsou reprezentovány entitou **Koalice**.

- **Zasedání**
  Záznam jednoho zasedání konaného během daného funkčního období. Má pořadové číslo, datum konání a účast vyjádřenou počtem přítomných členů. Přítomnost jednotlivých členů je zaznamenávána pomocí **BylPřítomen**. Možnosti přítomnosti (přítomnost, částečná přítomnost, nepřítomnost) určuje enumerátor **Docházka**. K uložení odkazu na záznam zasedání lze použít **ZasedáníZáznam**.

- **Hlasování**
  Jedno z hlasování provedených v daném zasedání. Entita mimo jiné obsahuje čas konání, předmět hlasování a součty jednotlivých hlasovacích možností. Výsledek hlasování je odkazem na enumerátor **VýsledekHlasování**, jež momentálně nabízí možnosti *přijato* a *nepřijato*.

- **HlasováníČlena**
  Reprezentuje hlas člena udělený při konkrétním hlasování. Hlas může nabývat hodnot z enumerátoru **HlasovacíMožnost**. Aktuální hlasovací možnosti jsou *ano*, *ne*, *zdržel/a se*, *nehlasoval/a*, *omluven/a* a *tajná*.

4.4.3 Objektově relační mapování

V systému je pro práci s relačními databázemi využíváno objektově relačního mapování (ORM) s využitím knihovny jazyka Python nazvané `SQLAlchemy`[5]. V balíčku `api.orm` jsou defino-

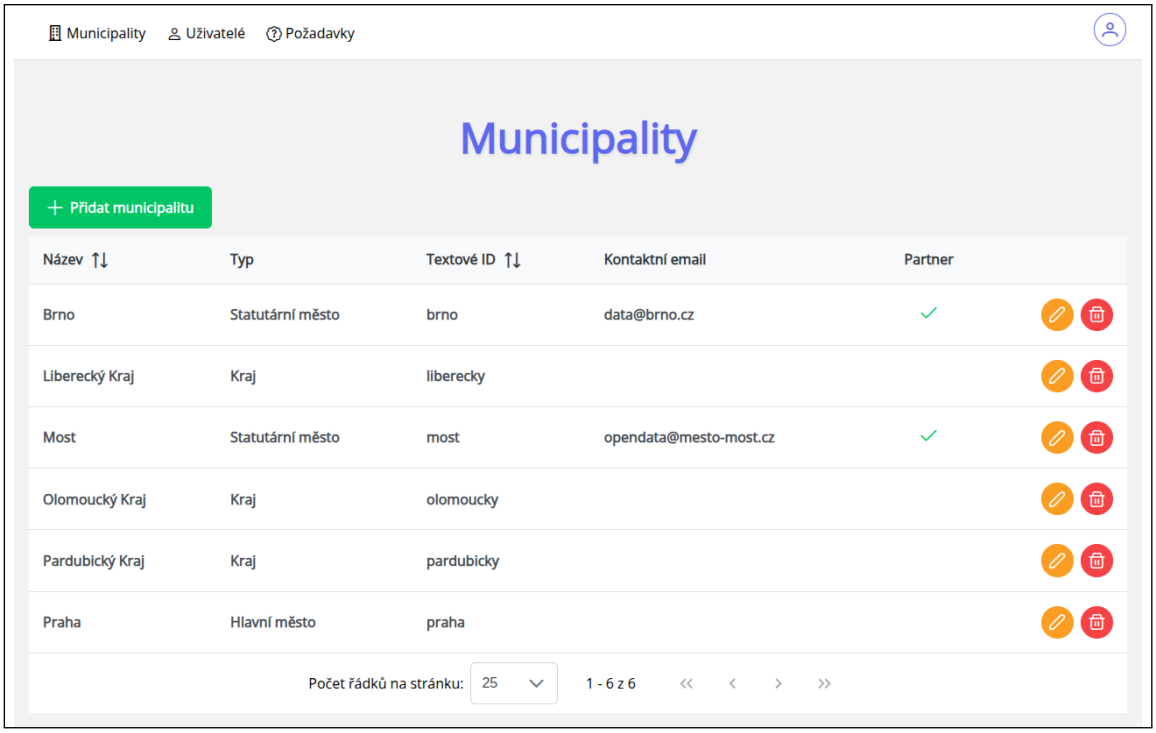
[5] https://sqlalchemy.org/

vána schémata objektů zastupujících jednotlivé databázové entity. Tomu odpovídá také pojmenování jejich atributů, které spolu s nastavenými omezeními zrcadlí sloupce a omezení obou používaných databázových schémat

Pro komunikaci s databází je implementována třída `ORMHandler`, která skrze knihovnu `SQLAlchemy` vrací objekt typu `Session`, jímž je možné ukládat změny databázových objektů. Třída nabízí separátní metody pro vytvoření objektu `Session` pro centrální databázi, nebo pro libovolnou orgánovou databázi, jež je nalezena dle textových identifikátorů municipality a orgánu předaných parametry při volání. Údaje pro připojení k databázovému serveru třída načítá z proměnných definovaných v prostředí, nebo používá výchozí hodnoty, pokud potřebné proměnné nejsou v prostředí nalezeny.

4.5 Uživatelské rozhraní

Tato sekce se vzhledem k povaze zadání zaměřuje pouze na administrační úsek uživatelského rozhraní. Po přihlášení do administrace se uživatel ocitá na stránce se seznamem municipalit (viz obrázek 4.6), z něhož jednotlivé municipality může spravovat barevně značenými akčními tlačítky.



Obrázek 4.6: Snímek obrazovky ukazující seznam v systému evidovaných municipalit. Barevně značená akční tlačítka slouží k jejich správě. Kliknutím na řádek municipality v seznamu je uživatel přesměrován na výběr funkčních období zastupitelstva municipality. Na snímku je vyobrazen pohled globálního administrátora, který v horním navigační liště má navíc přístup k seznamu uživatelů a uživatelským požadavkům na správu nebo vytvoření municipality. Seznam municipalit byl pro účely demonstrace zkrácen.

Po výběru konkrétní municipality je uživatel přesměrován na seznam funkčních období zastupitelstva municipality. Výběr jiného typu zastupitelstva v rozhraní chybí. Přes volbu

funkčního období a zasedání se lze proklikat až ke konkrétnímu hlasování se seznamem hlasů jednotlivých členů zastupitelstva (snímek umístěn do přílohy A.1). Všechny zmíněné entity od funkčního období až po hlasy členů je možné v rozhraní upravovat. Odkazy na horní liště pak směřují k možnostem správy politických subjektů a informací o členech zastupitelstva.

Pro přidání nové municipality (dostupné pouze globálním administrátorům), pro podání žádosti o přidání (dostupné naopak běžným registrovaným členům) či pro úpravu existující municipality existuje formulář, k němuž je napevno připojeno také vytvoření, respektive úprava přidruženého zastupitelstva (viz příloha A.2). Volba jiného typu orgánu není možná.

4.6 Požadavky na rozšíření

Praktickým výstupem této práce je rozšíření administračního modulu vytvořeného zároveň s diplomovou prací [30]. Rozsah plánovaného rozšíření je popsán touto sekcí. Požadavky byly konzultovány s týmem projektu Zastupko.cz a odrážejí jejich přání i přání některých zájemců o využívání systému k vlastnoruční správě dat, která byla s týmem komunikována.

4.6.1 Zobecnění orgánů municipalit

Ačkoliv jsou některé interní metody backendu připraveny k práci s různými typy politických orgánů, jejich nemalá část napevno počítá pouze se zastupitelstvy. Jak již bylo naznačeno v sekci 4.2, ani frontend nedisponuje plnou podporou, respektive i přes to, že jeho veřejný modul je již plně generalizován, v administračním modulu není možné vytvářet nebo i pouze nahlížet do orgánů jiného typu a každá municipalita je zde pevně spjata se zastupitelstvem. Úkolem tohoto bodu bude dát administrátorům a správcům municipalit možnost pod municipalitami vytvářet a upravovat další typy orgánů tak, jak je tomu možné v případě zastupitelstev.

4.6.2 Databázové operace

Aktuálně je možné provádět správu databází pouze z databázového serveru interakcí například pomocí přímých SQL příkazů nebo s využití grafických rozhraní jako *phpMyAdmin* či *Adminer*. Cílem tohoto bodu je přenést určité funkce správy přímo do stávajícího administračního rozhraní. Databázové rozšíření by mělo podporovat následující funkce:

- **Zobrazení seznamu**
  Nové rozhraní by mělo nabízet možnost zobrazit seznam databází, které aplikace používá, a to včetně jejich typu (centrální a orgánová) a data poslední změny.
- **Export**
  Databáze by mělo jít exportovat do SQL skriptu a stáhnout přes internetový prohlížeč do zařízení uživatele.
- **Import**
  Rozhraní by mělo podporovat provádění SQL skriptů nahraných uživateli. Při tomto importu by aplikace sama měla rozpoznat, do jakých databází skript zasahoval. Tyto informace by měla aplikace využít k automatickým aktualizacím seznamu databází zmíněného výše, a to nejen při aktualizacích, ale i v případě přidání nových nebo smazání stávajících databází.

- **Zálohování**
  Rozhraní by mělo dovolovat ruční vytvoření zálohy vybrané databáze. Určité akce by mohly spouštět automatické zálohování. Zálohy by měly být uloženy na serveru a dostupné ke stažení všem administrátorům. Také u databází, které již byly smazány, by přístup k případným dříve vytvořeným zálohám měl být nadále otevřený.

- **Správa**
  Správou je myšleno vytváření, přejmenování a mazání databází přímo v grafickém rozhraní. Všechny operace budou sloužit především k testovacím účelům, kde může nastávat častější potřeba databázových manipulací. Při vytváření databází by měl být k dispozici výběr z předem daných typů, na základě nichž dojde k inicializaci databáze dle odpovídajícího schématu. Pro předejití nehodám při mazání databází by odstranění mělo být prováděno bezpečným způsobem, který znemožní úplnou ztrátu dat. Všechny případné dřívější zálohy smazaných databází by rovněž měly zůstat přístupné.

Vzhledem k citlivosti těchto operací jsou funkce orientovány pouze na potřeby administrátorů, kteří by měli mít možnost spravovat databáze odkudkoliv bez přístupu k databázovému serveru. Běžné účty by neměly mít možnost se k databázovým operacím jakkoliv dostat. Předpokládá se tedy dostatečné proškolení a technická zdatnost. Funkce by měly být navrženy se zřetelem na minimalizaci rizika destruktivních operací v případě neoprávněného přístupu k administrátorskému účtu.

4.6.3 Dílčí úpravy

Následující úpravy by měly být přidány pro celkové zjednodušení práce s administračním modulem. Jedná se o menší zásahy spadající do různých kategorií, které by měly dopomoci k lepšímu dojmu z používání systému a urychlit pracovní postupy:

- **Sledování historie úprav**
  Pro zvýšení transparentnosti manuálních zásahů existuje požadavek zaznamenávat a zobrazovat v administrátorském rozhraní historii úprav hlasování po jejich prvotním vložení. V reflexi na tento požadavek byly entity *Zasedání*, *Hlasování* a *HlasováníČlena* zvoleny jako nejvhodnější kandidáti, u nichž by měly být zaznamenávány úpravy. Funkcionalita by však měla být implementována dostatečně obecně, aby byla v budoucnu snadno rozšiřitelná o případné další entity. Pokud by záznamy z úprav byly vytvářeny v rámci všech municipalit, tj. i těch, jejichž správci o sledování záznamů nemají zájem, docházelo by ke zbytečnému zabírání místa na disku. Funkcionalita tedy by měla být nasazena volitelně pro specifické municipality.

- **Odkazy na sousední hlasování**
  Při sekvenčních manuálních úpravách hlasování patřících pod jedno zasedání je pro přechod na následující hlasování nutné vrátit se na předchozí stránku se seznamem všech hlasování a požadovaný záznam v něm vyhledat. Práci by urychlily odkazy na předchozí a následující hlasování přímo na stránce editace.

- **OpenAPI specifikace**
  Pro zjednodušení práce celého týmu s aplikačním rozhraním a přehlednější dokumentaci by bylo vhodné API endpointy popsat pomocí OpenAPI specifikace[6].

- **Dynamické načítání informací**
  Některé informace o municipalitách a jejich orgánech jsou uložené ve statických konfiguračních souborech ve formátu JSON. Tyto informace by se měly načítat dynamicky přímo z databáze pomocí nového endpointu.

[6] OpenAPI specifikace slouží k definování standardního, strojově čitelného aplikačního rozhraní. Více na webu OpenAPI Initiative: https://openapis.org/.

Kapitola 5

Návrh

Kapitola se zabývá návrhem rozšíření administračního modulu o funkce definované požadavky v sekci 4.6. Změny bude potřebné provést ve všech hlavních vrstvách systému, podle nichž jsou následující sekce členěny. Návrh tedy počítá se zásahy do databází i do administrátorského modulu serveru a klienta.

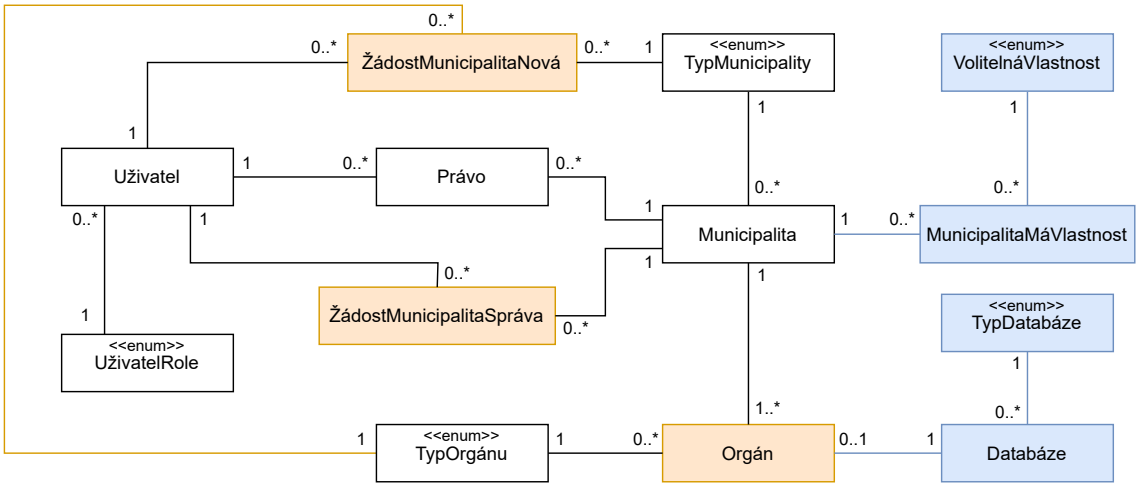
5.1 Databázová část

Práce bude rozšiřovat stávající datový model navržený v pracích [30, 57], jehož aktuální stav byl podrobněji analyzovaný v sekci 4.4. Následující text popisuje zásahy do tohoto modelu, jenž bude muset nově zahrnovat podrobnější informace o některých entitách a zároveň bude doplněn o entity nové, reflektující plánované úpravy. V závěru je zmíněno též rozšíření architektury o nový typ databáze sloužící k ukládání záloh.

5.1.1 Změny ve schématu centrální databáze

Aby bylo možné efektivně spravovat seznam databází a obsluhovat některé nové plánované funkce vyžadující ukládání metadat o municipalitách, budou v centrální databázi vytvořeny nové databázové entity zvýrazněné na obrázku 5.1 modrou barvou. Datový model bude doplněn následovně:

- **TypDatabáze** – Výčet typů databází, jež jsou v aplikaci používány, tj. centrální a orgánová s potenciálem budoucího rozšíření počtu typů.
- **Databáze** – Obsahem budou metadata o databázích, které aplikace používá. Přítomny budou záznamy jak o centrální, tak o všech databázích orgánů. Mezi ukládané atributy bude patřit název databáze, její typ, časové razítko poslední provedené změny obsažených dat a časové razítko poslední zálohy. Zaznamenáván by měl být taktéž příznak, zda databáze byla fyzicky smazána.
- **VolitelnáVlastnost** – Výčtová tabulka ukládající typy nepovinných vlastností nebo funkcionalit, jež administrátoři a správci municipalit mohou pro jednotlivé municipality volitelně aktivovat i deaktivovat. Jedinou volitelnou vlastností bude v této době sledování historie úprav, jak bylo popsáno v sekci 4.6. Tento návrh však počítá s budoucími rozšířeními v podobě dalších nepovinných vlastností.
- **MunicipalitaMáVlastnost** – Vazební tabulka mezi municipalitami a vlastnostmi určující, které volitelné vlastnosti byly pro jakou municipalitu aktivovány.



Obrázek 5.1: Schéma entit centrální databáze rozšířené o entity, které budou přidány jako součást této práce. Nové entity a vazby jsou vyznačeny modrou barvou. Oranžová barva značí zásahy do již existujících entit.

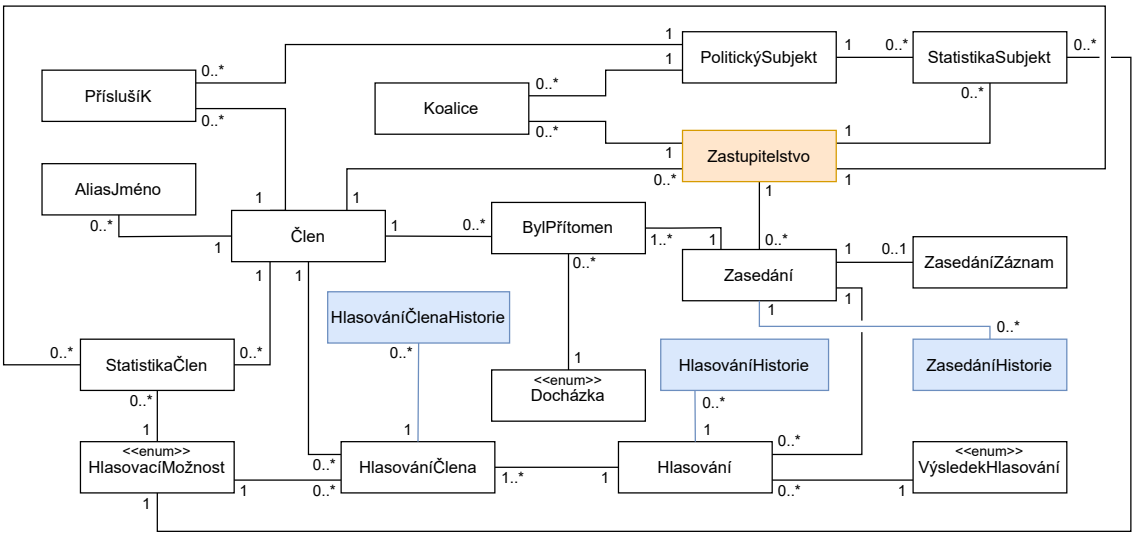
Dále jsou na obrázku 5.1 oranžově vyobrazeny již dříve existující entity, které budou upraveny přidáním nových atributů pro potřeby generalizace práce s politickými orgány a eliminace povinné návaznosti municipalit na zastupitelstva:

- **ŽádostMunicipalitaNová** – Záznam žádosti o vytvoření nové municipality nyní bude rozšířen o typ orgánu, který má být spolu s municipalitou vytvořen, namísto předpokladu, že se má jednat o zastupitelstvo.
- **Orgán** – S plánovaným rozšířením povolujícím vytváření nových orgánů by také mělo být implementováno jejich mazání. Do této tabulky bude z toho důvodu přidán příznak značící smazání daného orgánu. Ve skutečnosti podobně jako u smazaných municipalit půjde o pouhé zneviditelnění a záznam z bezpečnostních důvodů i kvůli prevenci omylů zůstane nadále dostupný v databázi.

5.1.2 Změny ve schématu databáze orgánu

Schéma databází orgánů rovněž musí projít změnami. V plánu je vytvoření nových entit k uchovávání historie manuálních úprav vybraných záznamů a také rozšíření stávající entity o další atributy, které umožní zaznamenávání nových informací. Nové entity jsou na obrázku 5.2, podobně jako v případě schématu centrální databáze, odlišeny modrou barvou a úpravy stávajících oranžovou:

- **HlasováníHistorie**, **HlasováníČlenaHistorie**, **ZasedáníHistorie** – Tyto tabulky budou sloužit k ukládání historie ručních úprav databázových záznamů patřících pod entity *Zasedání*, *Hlasování* a *HlasováníČlena*. Bližší popis návrhu a způsobu fungování těchto tabulek je v následující podsekci.
- **Zastupitelstvo** – Přidány budou atributy značící úplnost datasetu pro dané volební období a časové razítko poslední úpravy datasetu. Tyto informace jsou nyní obsažené ve statických konfiguračních souborech, ale v databázi chybí.



Obrázek 5.2: Rozšířené schéma entit orgánové databáze. Entity a vazby, jež budou jako součást této práce nově vytvořeny, jsou označeny modře, zatímco zásahy do již existujících entit jsou vybarveny oranžovou.

5.1.3 Formát tabulek historie úprav

V předchozí odstavcích bylo řečeno, že manuální úpravy dat entit *Hlasování*, *HlasováníČlena* a *Zasedání* budou zaznamenávány do nových tabulek historie, pokud k nim došlo v orgánech, které patří municipalitám, jež se tuto vlastnost rozhodly aktivovat. Každá ze sledovaných entit bude mít korespondující tabulku historie se sloupci kopírujícími svým názvem i datovým typem všechny atributy dané entity, jež jsou upravitelné v administračním panelu. Dále se bude historický záznam klíčem odkazovat na původní řádek a bude opatřen sloupci pro jméno autora úpravy a časové razítko, kdy úprava byla provedena. Větší počet řádků v tabulce historie může odkazovat na jeden řádek v původní tabulce – při každé manuální úpravě je vytvořen jeden historický záznam.

Příklad tabulky ukládající hlasování členů a jí odpovídající tabulky historie ukazuje obrázek 5.3, který zároveň naznačuje postup vytváření historických záznamů. V konkrétním příkladu z obrázku administrátor v grafickém rozhraní nejprve změnil uloženou hodnotu členem zastupitelstva zvolené hlasovací možnosti. Originální hodnota byla systémem s příslušným odkazem na původní záznam zapsána do tabulky historie. Administrátor pokračoval změnou dalších atributů. Pro hodnoty před změnou byl opět vytvořen nový historický záznam.

Uvedený postup dovoluje, aby současný stav byl vždy uložen v původní tabulce a do tabulek historie vůbec nemuselo být přistoupeno, pokud zobrazení aktuálních hodnot postačuje. To, že jsou ukládány pouze hodnoty, které byly opravdu změněny, a zbytek atributů zůstává prázdný, zase napomůže k šetření úložného prostoru[1]. Nevýhodou tohoto přístupu je složitější rekonstrukce časové posloupnosti úprav, jelikož pro získání stavů všech atributů v konkrétním bodě a jejich následujících hodnot nemusí stačit pouze porovnat daný historický záznam s dalším v řadě. Předpokládá se, že žádný individuální záznam nebude upravován tak často, aby zpracování jeho historických záznamů složitějším algoritmem znamenalo

[1]Entita *HlasováníČlena* má malý počet atributů a jako příklad byla použita pro svou jednoduchost. Větší entity jako *Hlasování* skýtají výraznější potenciál k ušetření prostoru v rámci jejich tabulek historie.

Obrázek 5.3: Princip ukládání historie změn záznamů sledovaných entit. Tabulky historie kopírují atributy upravitelné v grafickém rozhraní. Při úpravě takového atributu je předchozí hodnota před změnou uložena do historického záznamu. Nezměněné hodnoty se kvůli úspoře úložného prostoru neukládají. V obou vyobrazených tabulkách byly pro přehlednost zanedbány záznamy související s jinými hlasováními.

problém. Algoritmus pro rekonstrukci posloupnosti bude prováděn v backendu aplikace a proto je více přiblížen v serverové sekci této kapitoly společně s podobou jeho výstupu.

5.1.4 Databáze záloh

Pro realizaci ukládání databázových záloh bude ke stávající architektuře přidán další databázový server. Tato separace snižuje riziko ztráty všech dat při chybě v primárním úložišti. Předpokládá se, že zálohy budou objemná data bez jednotné struktury, která po zápisu budou čtena jen zřídkakdy, vyhledávání v nich bude probíhat primárně pomocí indexovaného[2] identifikátoru databáze a již uložené záznamy nebudou nikdy upravovány. Pro rozsáhlá data tohoto typu bude zvolen databázový systém typu NoSQL, který by zároveň měl nabídnout možnost snadného škálování, bude-li potřeba [20]. Pokud budoucnost přinese potřebu rozšíření ukládaných informací, bezschématový model NoSQL dovolí ukládat spolu se zálohami jakákoliv nová metadata bez nutnosti potenciálně citlivé práce se zálohovanými daty při migracích na nové schéma. Zálohy budou ukládány v podobě dokumentů, které by měly obsahovat minimálně následující atributy:

- `db_id` – indexovaný identifikátor shodný s identifikátorem záznamu o databázi v centrální DB (viz podsekce 5.1.1), podle nějž bude záloha přiřazena ke správné databázi;
- `backup_file` – celá záloha v binární podobě;
- `backup_timestamp` – časové razítko, kdy byla záloha provedena;

[2]Databázové indexy jsou datové struktury umožňující rychlejší vyhledávání. Detailnější informace například na https://mongodb.com/resources/basics/databases/database-index

- `backup_trigger` – informace o tom, jak byla záloha spuštěna (manuálně nebo automaticky při jiné akci).

K vyhledání celého seznamu záloh bude využíván atribut `db_id`. Pokud bude dodržen uvedený formát dokumentu, vyhledávání bude možné také podle časového razítka nebo spouštěče zálohy, jenž by měl kvůli tomuto účelu nabývat pouze předvídatelných, předem definovaných hodnot. Nepředpokládá se nutnost podle těchto dodatečných atributů vyhledávat často, proto pro ně nebude v databázi vytvořen index.

5.2 Serverová část

Tato sekce se věnuje popisu změn v serverové části administračního modulu. Jsou zde definovány rozšiřující endpointy pokrývající nové požadované funkcionality. Představen je algoritmus navržený k procházení historických záznamů manuálních zásahů do dat nebo algoritmus dohlížející na správnou aktualizaci metadat o databázích při importování skriptů v jazyce SQL.

5.2.1 Koncové body

Stávající aplikační rozhraní administrátorského modulu bude kvůli novým potřebám rozšířeno o následující endpointy:

- `/flask/admin/body-type`
- `/flask/admin/municipality/<string:municipality_id>/body`
- `/flask/admin/municipality/<string:municipality_id>/<string:body_type>`
- `/flask/admin/database-type`
- `/flask/admin/database`
- `/flask/admin/database/<int:database_id>`
- `/flask/admin/database/<int:database_id>/export`
- `/flask/admin/database/<int:database_id>/backup`
- `/flask/admin/database/<int:database_id>/<int:backup_id>`
- `/flask/admin/database/import`

První trojice jmenovaných endpointů bude přidána kvůli zobecnění práce s orgány municipalit a sloužit hlavně pro úpravy existujících a vytváření nových. Každá municipalita může disponovat pouze jedním orgánem stejného typu, proto je kombinace identifikátoru municipality a typu orgánu dostatečná k nalezení požadovaného orgánu.

Následující uvedené endpointy budou sloužit k manipulacím s databázemi. K identifikaci databáze bude použit její číselný identifikátor, který databáze po svém vytvoření dostane při zapsání záznamu o její existenci do centrální DB. Přestože obvykle v databázových systémech musí být jména databází unikátní, nebylo by vhodné je použít jako identifikátor pro práci s představenými endpointy. Nemusí totiž být vždy unikátní v rámci samotného seznamu databází uloženého v centrální DB, protože po smazaných databázích

nadále zůstávají záznamy – byť s patřičně nastaveným příznakem –, které původní jména stále uchovávají. S těmito záznamy půjdou z rozhraní provádět operace jako načítání záloh patřících k dříve smazaným databázím, proto je jejich ponechání žádoucí. Oproti tomu číselný identifikátor bude zároveň primárním klíčem tabulky a jeho jednoznačnost tedy bude zaručena.

Žádný z nových endpointů nebude přístupný pro neregistrované uživatele. První trojice koncových bodů bude dostupná pro globální administrátory a registrované uživatele s povolením upravovat v URL definovanou municipalitu. Databázové endpointy pak požadavky účtů bez práv globálního administrátora nebudou přijímat vůbec. Případná data budou v požadavcích i v odpovědích přidaných endpointů přenášena v objektech odpovídajících formátu JSON. Výjimkou budou koncové body pro stažení zálohy a nahrání importovacího skriptu. Připravené zálohy budou zasílány v HTTP odpovědích jako binární soubor. Obdobně budou v HTTP požadavcích přenášeny skripty určené k importu.

5.2.2 Import databázových skriptů

Importování dat do databází, případně kompletní vytváření nových databází či jejich mazání, bude moci provádět nahráním skriptu v jazyce SQL, který server následně provede. Tato funkcionalita, stejně jako další práce s databázemi, bude omezena pouze pro účty s právy globálního administrátora, u nichž se počítá s dostatečnými znalostmi a zodpovědným přístupem při jejím používání.

Výše bylo v podsekci 5.1.1 popsáno, že aplikace bude nově ve své centrální databázi udržovat záznamy s metadaty o všech databázích, s nimiž pracuje. Kromě toho je kvůli minimalizaci rizik spojených s významnými zásahy do databází žádoucí, aby ty databáze, jež mají být při importech jakkoliv zasaženy nahranými skripty, byly před aplikováním změn zálohovány. Z tohoto důvodu není možné nahrané skripty pouze přečíst a spustit bez toho, aby aplikace zkoumala ve skriptu obsažené operace.

Jako SQL příkazy, které je nutné při procházení skriptu rozeznávat a přizpůsobovat podle nich běh algoritmu pro import, byly identifikovány `USE`, `CREATE DATABASE` a `DROP DATABASE`, tedy případy, kdy se k databázi přistupuje před jejími dalšími úpravami nebo když je databáze vytvářena či odstraňována. V průběhu vykonávání skriptu budou předběžně vytvářeny zálohy dotčených databází před každým provedením operací `USE` a `DROP DATABASE`. Interně bude rovněž zaznamenáváno pořadí, v němž jsou všechny sledované operace pro každou konkrétní databázi prováděny.

Tabulka 5.1 nabízí přehled bodů, které by měly platit po každé uvedené databázové operaci. Výjimka nastává, pokud se v rámci jedné databáze objeví ihned po sobě operace `DROP DATABASE` a `CREATE DATABASE`. Tato sekvence je z pohledu algoritmu prakticky ekvivalentní s operací `USE`, jelikož jejím výsledkem je pouze aktualizace existující databáze a z praktického hlediska zde nedochází ke smazání nebo vytvoření nové databáze. Algoritmus importu by měl tuto situaci rozpoznat a nevytvářet při této sekvenci nový záznam o databázi v centrální DB, ale pouze aktualizovat původní.

Algoritmus by měl po jednom procházet a provádět příkazy z uživatelem nahraného skriptu jazyka SQL. U každého příkazu bude nutné nejprve kontrolovat, zda nejde o operaci uvedenou v tabulce 5.1. Pokud ano, bude třeba interně tuto skutečnost poznamenat i se jménem databáze, jíž se týká, případně ještě před spuštěním operace také vytvořit zálohu databáze, pokud je tak v tabulce uvedeno. Chyba v průběhu provádění SQL příkazu nebo dosažení konce skriptu by mělo spustit druhou fázi. V ní budou čteny uložené operace

a aktualizovány záznamy o databázích v centrální DB, aby odpovídaly reálnému stavu dotyčných databází po provedení skriptu.

Druhá fáze algoritmu musí být spuštěna i v případě chyby při provádění kódu jazyka SQL z důvodu, že příkazy úspěšně dokončené před samotným výskytem chyby, které ovlivnily schéma nebo existenci databází, není možné jednoduše vrátit zpět. Jejich konsekvence je vždy nutné promítnout do metadat.

Operace	Výsledek
`DROP DATABASE`	– záloha uložena – databáze smazána – v centrální DB nastaven příznak smazání databáze
`CREATE DATABASE`	– databáze vytvořena – nový záznam o databázi uložen v centrální DB – aktualizováno razítko poslední změny databáze
`USE`	– záloha uložena – aktualizováno časové razítko poslední změny databáze
`DROP DATABASE` následováno `CREATE DATABASE`	– záloha uložena – aktualizováno časové razítko poslední změny databáze

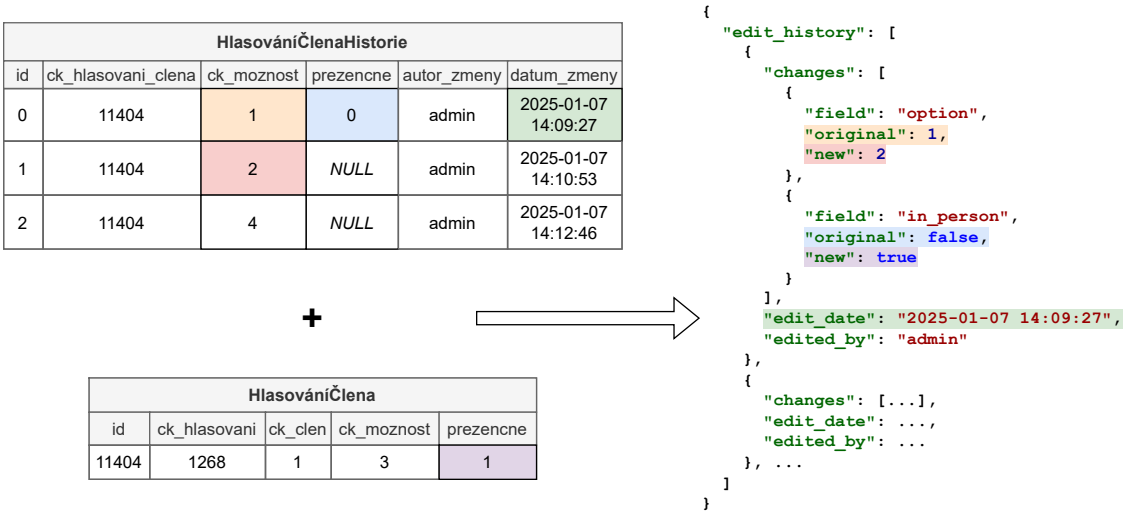
Tabulka 5.1: Seznam databázových operací a akcí, které by se měly po těchto operacích nad danou databází provést. Pokud se objeví sekvence příkazů `DROP DATABASE` a `CREATE DATABASE` provedených v rámci jedné databáze hned za sebou, jedná se o speciální případ, jehož se samostatně definované řádky pro obě operace netýkají.

5.2.3 Rekonstrukce časové posloupnosti úprav

Z tabulek historie, jejichž formát byl definován podsekcí 5.1.3, je pro jednodušší manipulaci třeba sestavit přesnou posloupnost ve strukturovaném formátu. Posloupnost by jasně měla uvádět nejen původní hodnotu v době editace danou časovým razítkem záznamu v tabulce historie, ale také hodnotu, jakou byla nahrazena, která se z důvodu popsaných v uvedené podsekci nemusí vždy nacházet hned v následujícím záznamu.

K sestavení kompletní posloupnosti pro daný řádek sledované databázové entity bude sloužit algoritmus, který bude dostatečně obecný, aby šel aplikovat na entity s libovolným počtem atributů, jejichž historie je zaznamenávána. Algoritmus bude implementován tak, aby posloupnost rekonstruoval do objektu formátu JSON, jenž bude jednoduše zpracovatelný frontendem aplikace. Na obrázku 5.4 je vidět struktura objektu, který pro každé časové razítko bude obsahovat seznam změněných atributů s jejich původní a následující hodnotou. Pokud nějaký atribut nebyl v rámci určitého záznamu upraven, pro tento záznam bude z objektu vynechán.

Při rekonstrukci historie pro daný záznam sledované entity bude třeba procházet všechny příslušné historické záznamy a u každého hledat první neprázdný atribut, jehož hodnota se uloží jako původní. V navazujících historických záznamech potom bude hledána hodnota, na niž byla původní změněna. Pokud nová hodnota nebude nalezena v žádném z pozdějších záznamů, znamená to, že od poslední změny nebyl atribut dále upravován a bude použita aktuální hodnota z aktuálního databázového záznamu. Algoritmus tento postup bude opakovat pro každý neprázdný atribut všech historických záznamů spojených s aktuálním databázovým záznamem.



Obrázek 5.4: Ukázka rekonstrukce historie úprav do strukturovaného objektu formátu JSON obsahujícího pro každý záznam v tabulce historie původní i novou hodnotu, která se může nacházet i v jakémkoliv následujícím historickém záznamu nebo dokonce v původní tabulce. Ve výsledném objektu byly kvůli stručnosti další položky zkráceny. Barvy zvýrazňují z jakých míst tabulek byly převzaty hodnoty ve výsledném JSONu.

5.3 Klientská část

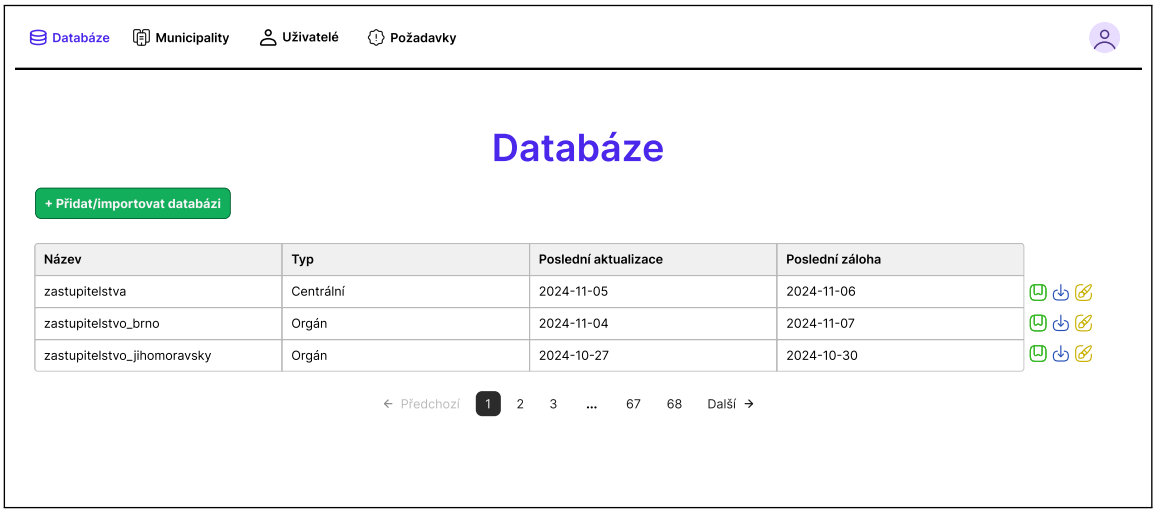
Klientská část administračního systému není určena běžnému návštěvníkovi webu. Používána je především technicky zdatnými nebo proškolenými uživateli primárně na osobních počítačích a laptopech, proto není třeba klást takový důraz na responzivitu jako ve veřejné části aplikace. Přesto byla při návrhu rozšíření stávajícího rozhraní vyvíjena snaha o jistou úroveň uživatelské intuitivnosti. Rozšířené rozhraní se snaží držet zavedeného vzhledu a návrhy byly diskutovány a upravovány na základě zpětné vazby od uživatelů existujícího administračního rozhraní.

Na obrázku 5.5 je vyobrazen návrh nového seznamu databází užívaných v systému. V navigační liště vedle odkazu vedoucího na přehled municipalit je umístěn nový odkaz, jenž byl k přesměrování na novou stránku vytvořen a který se nebude vykreslovat běžným účtům, jelikož manipulace databází je zamýšlena jako striktně administrátorská operace. Barevně značená akční tlačítka budou sloužit k exportu databáze do SQL souboru (modré), úpravám atributů databáze (žluté) a zobrazení dostupných záloh (zelené). Smazání databáze z rozhraní je možné, ale na rozdíl od již existujících přehledů je umístěno až za proklikem na žluté tlačítko jako jedna z bariér k zabránění neúmyslnému smazání[3].

Tlačítko k přidání nové databáze rovněž viditelné na obrázku 5.5 přesměruje uživatele na novou stránku, jejíž návrh ukazuje obrázek 5.6. Zde si uživatel bude moci vybrat mezi importováním databáze nahráním SQL souboru nebo vytvořením prázdné databáze s jedním z předem určených schémat. Výhoda prvního jmenovaného způsobu spočívá v možnosti nahrání databáze společně s daty.

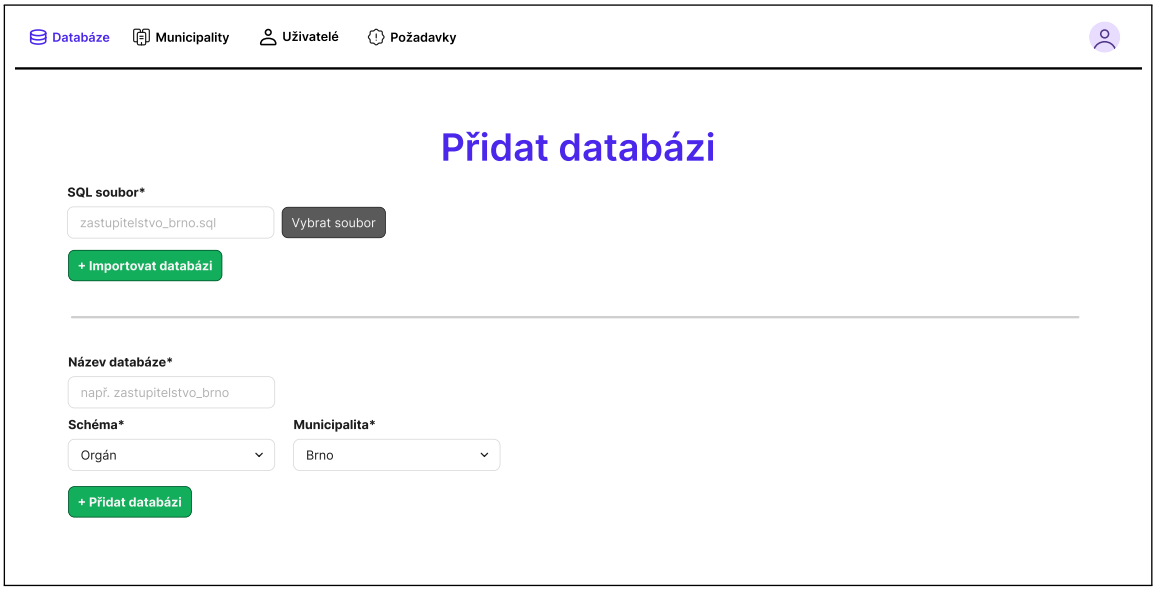
Obrázek 5.7 představuje rozšířený seznam municipalit. Původní stránka nenabízí možnost spravovat jiný orgán než zastupitelstvo. Nový návrh zavádí ke každé municipalitě roz-

[3]Všechny databáze však budou před odstraněním automaticky zálohovány, takže k nenávratné ztrátě dat by nemělo dojít ani při nešťastném smazání nezamýšlené databáze.

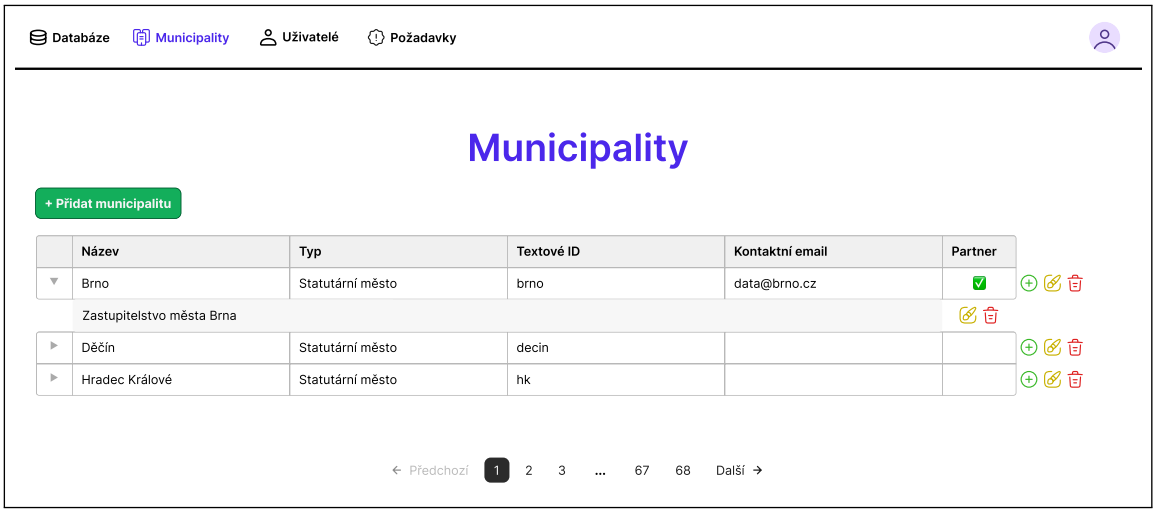


Obrázek 5.5: Návrh databázového seznamu. Zobrazuje seznam všech v systému používaných databází. Každou lze individuálně spravovat, exportovat nebo zálohovat akčními tlačítky. Tlačítko nad tabulkou přesměruje na novou stránku s možnostmi vytvoření nové DB.

balovací seznam podřízených orgánů. Rozbalení seznamu přidává oproti původnímu řešení jedno kliknutí myší navíc předtím, než uživatel může být přesměrován ke správě orgánu. Navržené řešení bylo zvoleno jako ústupek zachovávající kompaktnost seznamu, ale pokud čas ukáže, že jsou uživatelé s přidaným kliknutím nespokojeni, další možností je implicitně celý seznam rozbalit automaticky. Funkce tlačítka pro přidání nové municipality, tedy přesměrování na novou stránku s formulářem k tvorbě municipality, zůstává nezměněna. Ke žlutě a červeně značeným akčním tlačítkům u municipalit sloužícím k jejich úpravám a mazání návrh přidává zelené tlačítko, které po kliknutí otevře modální okno s možností založení nového orgánu dané municipality. Podobnými barevnými tlačítky disponují také orgány. Žluté a červené jsou obdobou svých protějšků u municipalit. Tlačítka k mazání municipalit i orgánů budou přístupná pouze administrátorům.



Obrázek 5.6: Návrh stránky pro přidání nové databáze. Uživateli je prezentována možnost importování z SQL souboru nebo vytvoření prázdné DB s vybraným schématem.



Obrázek 5.7: Návrh na rozšíření seznamu municipalit o orgány municipalit. Každá municipalita má nově rozbalovací seznam orgánů. Všechny municipality i orgány lze spravovat akčními tlačítky.

Kapitola 6

Implementace

Klíčovými cíli implementace bylo rozšířit původní administrační modul o správu různých typů politických orgánů a umožnit globálním administrátorům spravovat databáze rovnou z aplikace bez nutnosti přímého přístupu k databázovému serveru. Tato kapitola popisuje konkrétní kroky vedoucí k realizaci těchto i dalších změn podle návrhu popsaného v kapitole 5. Uvedení některých řešených problémů by neposkytlo dostatečnou hodnotu. Takové byly v této kapitole zanedbány. Výsledný praktický výstup však realizoval všechny vytyčené požadavky.

Protože implementace vychází z již existujícího administračního systému, který byl výstupem práce [30], byla do velké míry omezena následujícími hlavními technologiemi, na kterých byly stavěny hlavní komponenty systému:

- **Flask** – framework v jazyce Python, v němž je implementován backend aplikace;
- **React** – knihovna jazyka JavaScript, jež byla použita pro uživatelské rozhraní;
- **MariaDB** – relační databázový systém uchovávající aplikační data.

Nové technologie a knihovny, které byly při rozšiřování aplikace zakomponovány ke stávajícím, jsou zmíněny v příslušných sekcích této kapitoly.

6.1 Modul pro správu databází

Jádrem této práce jsou nové databázové operace dostupné globálním administrátorům, které byly představeny v požadavcích v sekci 4.6.2. Návrh klíčových změn byl dále popsán v kapitole 5. Následující text se věnuje zásadním aspektům rozšíření datového modelu a samotné implementace správy, exportování, zálohování a importování databází na úrovni serverové části aplikace i uživatelského rozhraní.

6.1.1 Rozšíření datového modelu

Databáze pojmenovaná určitým jménem, která je následně smazána a později nahrazena novou s identickým jménem, by měla být z pohledu systému vnímána jako jiná databáze s oddělenou množinou záloh. Proto je nutné jednotlivým databázím přiřazovat unikátní identifikátory nezávislé na jejich jméně, případně také o jednotlivých databázích ukládat metadata, která se v seznamu databází budou zobrazovat uživateli. To vedlo k rozšíření datového modelu centrální databáze podle návrhu v sekci 5.1.1 o entity **TypDatabáze**

a **Databáze**. První jmenovaná dle návrhu funguje jako výčet aktuálně v systému využívaných typů databází, tj. centrální a orgánová. Každý řádek entity **Databáze** pak představuje konkrétní databázi a disponuje těmito atributy:

- `id` – unikátní číselný identifikátor databáze;
- `nazev` – název databáze, který by měl být ve všech případech konzistentní s opravdovým pojmenováním databáze v databázovém systému;
- `ck_typ_databaze` – odkaz na řádek entity **TypDatabáze** určující schéma databáze, po inicializaci jde o neměnnou položku;
- `posledni_zmena` – časové razítko posledního zásahu do dat, které je automaticky aktualizováno při manuálních úpravách z grafického rozhraní i při úpravách skrze skripty jazyka SQL (více o možnosti nahrávání SQL skriptů dále v této sekci);
- `posledni_zaloha` – časové razítko poslední manuální či automatické zálohy;
- `smazano` – příznak udávající stav databáze, jeho nastavení říká, že databáze byla fyzicky smazána z hlavního databázového disku a zůstává o ní pouze záznam, případně také zálohy v databázi záloh (více o zálohování rovněž dále v této sekci).

Vzhledem k tomu, že změny bylo nutno provádět nad již existující databází, byl pro všechny úpravy vytvořen migrační skript v jazyce Python, který po spuštění s využitím knihovny `SQLAlchemy`[1] centrální databázi přenesl do stavu odpovídajícímu výše uvedenému popisu. Dále bylo třeba do centrální databáze zanést informace o všech aktuálně používaných databázích. Migrační skript k tomuto účelu automaticky využil pojmenovávací konvence databází orgánů popsané v sekci 4.4.2 a zaregistroval všechny databáze s prefixy `zastupitelstvo_` a `rada_`, které odpovídají všem momentálně rozeznávaným typům orgánů. Skript byl psán s kondicionálními klauzulemi v prováděných SQL příkazech, které zajišťují, že případným opětovným spuštěním migrace nad databází, jež dané změny již obsahuje, nedojde k chybám ani k přemazávání dat, která byla potenciálně do nových tabulek v mezičase uložena. Inicializační skript jazyka SQL pro centrální databázi byl upraven tak, aby provedené změny reflektoval a nová databáze vytvořena jeho prostřednictvím byla od začátku v aktuálním stavu.

6.1.2 Zobrazování a správa databází

Zobrazováním databází je myšlena schopnost systému prezentovat globálním administrátorům informace o databázích nově uložené v centrální DB přímo z administračního rozhraní. Správa databází je potom možnost databáze do tohoto seznamu přidávat, odebírat nebo upravovat již existující položky. Veškeré tyto akce jsou promítány do reálných databází umístěných na databázovém serveru, které záznamy v centrální DB zastupují.

Pro dané účely byla v aplikačním rozhraní vytvořena cesta s prefixem `/admin/database`, pod nímž jsou sdružovány všechny koncové body obsluhované třídami nově vzniklého modulu `Database`. Složitější metody, které jsou zároveň sdílené větším množstvím tříd obsluhy koncových bodů, byly po vzoru dříve definované architektury umístěny do balíčku `api.admin.service`. Komunikace s centrální databází je řešena objektově-relačním mapováním pomocí existující třídy `ORMHandler`, pro niž byla vytvořena nová schémata reprezentující rozšiřující databázové entity. Pokud koncový bod pro databázovou manipulaci

[1] https://sqlalchemy.org/

v požadavku očekává data (například při HTTP metodách `POST`), jsou validována pomocí schémat přidaných do balíčku `api.admin.requests`.

Koncové body pro samotné vracení seznamu databází a jejich správu byly drženy konzistentní s přístupem zavedeným v práci [30] a případná data vrací ve formátu JSON spolu se standardním stavovým kódem protokolu HTTP. Seznam databází je vracen jednoduchým čtením záznamů o databázích v centrální DB. U operací, které mění stav databází, je nejprve provedena samotná operace a až v případě bezchybného průběhu patřičná aktualizace záznamu o databázi. Konkrétně se jedná o následující:

- **vytvoření** – zřídí novou prázdnou databázi se zvoleným jménem a podle jednoho z předem vybraných schémat (centrální nebo orgánová), nakonec vytvoří nový záznam o databázi;
- **přejmenování** – přejmenuje databázi v databázovém systému a následně také v seznamu databází;
- **smazání** – kompletně odstraní databázi a nastaví příznak smazání v seznamu databází, před smazáním uloží zálohu, pokud je funkcionalita zapnuta (více o implementovaném způsobu zálohování níže).

Uživatelské rozhraní

Administrační část klientské aplikace napsané v knihovně `React` byla rozšířena o nové prvky. Pro správu databází byla úpravou inicializačního souboru přidána nová cesta `/admin/databaze`, u níž byl pomocí komponenty `ProtectedRoute` omezen přístup pouze pro uživatelské účty s rolí globálního administrátora. Na tuto cestu lze z uživatelského rozhraní přejít pomocí nového odkazu v navigační liště přidaného úpravou komponenty `AdminNavMenu`. Při vykreslování odkazu je kontrolována role přihlášeného uživatele, což opět zajišťuje jeho zobrazení pouze globálním administrátorům.

Nová cesta vede na komponentu `Databases`, která dotazem na backend načte seznam databází a zobrazí ho tak, jak je ukázáno na obrázku 6.1. Seznam o databázích vykresluje všechna uložená metadata a je primárně řazen podle typu databáze, sekundárně podle jejího názvu. Určité akce zmíněné dříve v této kapitole by měly být dostupné také pro smazané databáze, proto v rozhraní existuje zaškrtávací pole, který zobrazí a červeně zvýrazní také záznamy o již neexistujících. Takové záznamy jsou posílány v odpovědi z aplikačního rozhraní společně s ostatními hned při prvním požadavku a jejich odfiltrování nebo případné zobrazení je řešeno frontendem.

Oproti návrhu bylo přidání nové databáze a importování skriptu jazyka SQL rozděleno na dvě samostatné komponenty, kvůli čemuž by některá akční tlačítka směřovala na poměrně malý formulář. Místo přesměrování na novou stránku tedy bylo zvoleno zobrazení formuláře pro úpravy a vytváření databází v plovoucím okně s využitím komponenty `Dialog` z knihovny `PrimeReact`. Formulář byl implementován v komponentě `DatabaseDialog`, která své chování a podobu přizpůsobuje podle toho, zda jsou mu od komponentu `Databases` předány informace o databázi, jež má být upravena. Pokud ne, místo úprav je cílem vytvořit novou.

6.1.3 Exportování databází

Databázový modul kromě základní správy globálním administrátorům dále dovoluje vytvářet exporty databází pro přenos, vlastní zálohu nebo pro proces ladění chyb. Tato funkcio-

nalita byla do backendu přidána v podobě dalších dvou koncových bodů v dříve zmíněném modulu `Database`. Jeden koncový bod byl dedikován hromadnému exportu více databází, druhý pro samostatný export konkrétní databáze určené jejím identifikačním číslem v URL.

Vzhledem k tomu, že aplikace používá databázový systém MariaDB, byl pro vytváření exportů zvolen oficiální nástroj `mariadb-dump`, který ze zadané databáze umí vytvořit sadu příkazů jazyka SQL, jež po spuštění vytvoří ekvivalentní databázi. Použití nástroje bylo implementováno v metodě `export_db()` umístěné v balíčku `api.admin.services` s využitím standardního modulu jazyka Python nazvaného `subprocess`, jenž dovoluje aplikaci spustit další programy jako podřízené procesy. Nástroj `mariadb-dump` disponuje řadou přepínačů, které mohou ovlivnit finální podobu vytvořené sady příkazů, některé z nichž byly použity pro implementaci čtyř možností přizpůsobení exportu, jejichž libovolná kombinace může být navolena:

- `create_db` – uvede každou databázi příkazem `CREATE DATABASE`,
- `drop_db` – předznamená všechny případné příkazy `CREATE DATABASE` analogickým `DROP` příkazem,
- `drop_table` – předznamená všechny příkazy `CREATE TABLE` analogickým `DROP` příkazem,
- `include_data` – exportuje databázi také se všemi uloženými daty.

Volby těchto možností jsou přenášeny jako součást dat v požadavku na rozhraní. Server po přijetí požadavku na export vytvoří dočasný soubor, a skrze `export_db()` analyzuje požadované možnosti exportu a spustí nástroj `mariadb-dump` s potřebnými přepínači. Nástroj výsledek zapíše do dočasného souboru, který je následně serverem poslán zpět klientovi a pomocí vestavěného dekorátoru `after_this_request` je dočasný soubor vymazán z disku.

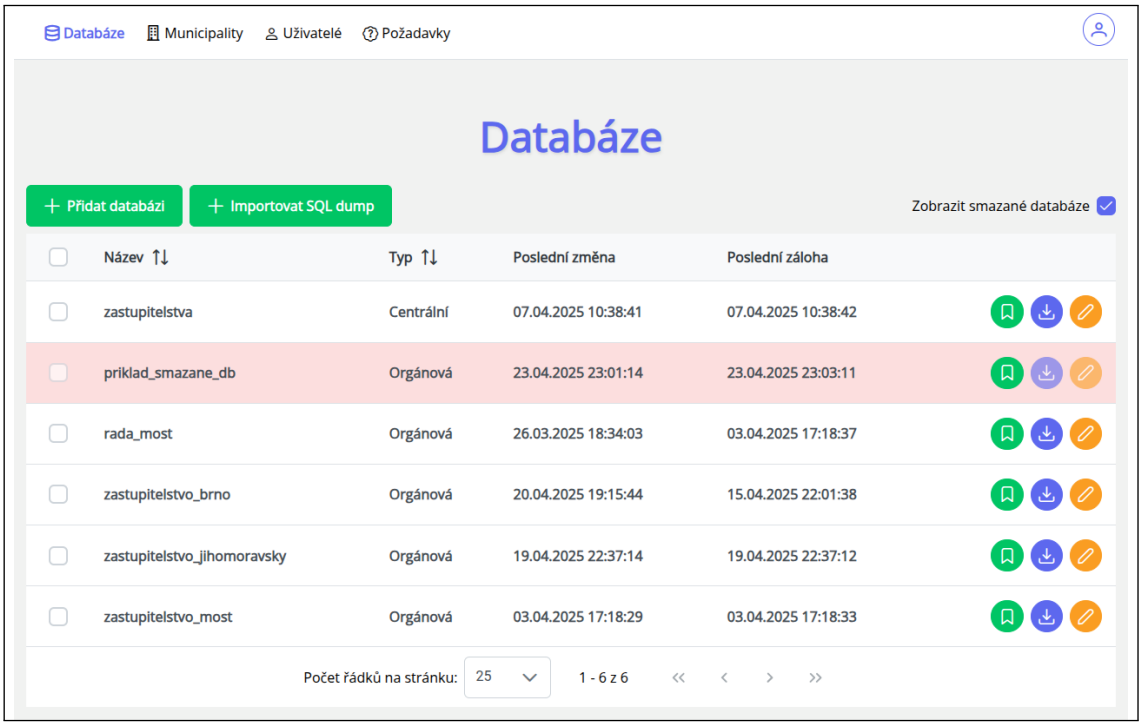
Uživatelské rozhraní

Ve frontendu je exportování přístupno ze seznamu databází, který je vidět ve výše představeném obrázku 6.1, a byla tedy implementována v téže komponentě `Databases`. Samostatné exporty je možné provádět přes modré akční tlačítko na řádku každé dané databáze. Po levé straně má naopak každý řádek zaškrtávací pole, které umožní označit více databází a následně všechny exportovat hromadně do jednoho souboru kliknutím na plovoucí tlačítko, které se po výběru více databází objeví. Označit všechny databáze zároveň lze zaškrtnutím pole v hlavičce tabulky. Kliknutí na jedno z tlačítek exportu vyvolá plovoucí okno implementované v komponentě `BackupDialog`, v němž je možné zaškrtáváním nastavit jednotlivé zmiňované možnosti exportu. Po dokončení exportu a obdržení souboru od backendu je na frontendu zavolána funkce `downloadFile()`, jež soubor stáhne přes internetový prohlížeč do zařízení uživatele. Pro smazané databáze je funkce exportu zakázána.

6.1.4 Zálohování databází

Sekce 5.1 podala důvody, proč k ukládání záloh bude použita nová databáze typu NoSQL. V průběhu implementace byla jako konkrétní technologie zvolena databáze MongoDB, která podporuje ukládání libovolně velkých binárních souborů[2], což se jeví jako vlastnost vhodná k ukládání záloh, které budou s časem stále více narůstat na objemu. MongoDB je

[2]Pomocí automatického rozdělování na menší kusy, více na https://mongodb.com/docs/manual/core/gridfs/.



Obrázek 6.1: Ukázka seznamu databází implementovaného v komponentě. Akčními tlačítky lze spravovat jednotlivé databáze. K zobrazení smazaných databází slouží zaškrtávací pole nad pravým horním rohem seznamu. Smazané databáze jsou vyznačeny červeně.

také v současnosti nejpopulárnější NoSQL databází[3], takže se dá předpokládat existence silné komunitní podpory a detailní dokumentace.

Databáze záloh byla implementována jako nepovinný modul. Aplikace se pokouší o připojení pouze za podmínky, že je environmentální proměnná `USE_MONGO` nastavena na hodnotu `1`. Na tento fakt je brán zřetel na všech místech backendu, kde je s touto databází komunikováno, a přístup při deaktivovaném MongoDB modulu je ošetřen způsobem vhodným pro danou situaci (například navrácením stavového kódu protokolu HTTP `403 Forbidden`). Při aktivaci modulu je konfigurace potřebná k připojení k MongoDB serveru načítána z proměnných prostředí, případně jsou použity výchozí hodnoty definované v balíčku `api.config`, pokud nejsou hodnoty v prostředí nastaveny.

Objektově dokumentové mapování datového modelu

Ke komunikaci s MongoDB serverem byla vytvořena třída `ODMHandler`, která při založení nové instance otevře připojení k databázi, jejíž jméno třídě bylo při inicializaci předáno. Podporováno je jak otevření trvalého globálního připojení, tak krátkodobého s použitím kontextového manažeru jazyka Python, kdy po vystoupení z kontextu připojení automaticky zaniká. Pomocí standardního modulu `atexit` je zajištěno, že také trvale otevřené připojení bude korektně uzavřeno i při nečekaném ukončení aplikace.

[3]K datu 1. 5. 2025 podle https://db-engines.com/en/ranking.

Třída `ODMHandler` používá knihovnu `mongoengine`[4], což je objektově dokumentový mapper postavený na driveru `PyMongo`[5] umožňující abstrahovat záznamy z databáze MongoDB jako objekty jazyka Python. K zajištění jednotného rozhraní pro práci s ukládanými dokumenty se zálohami byla definována třída `DatabaseBackup` dědící z knihovny `mongoengine`, jejíž atributy odpovídají těm, které byly navrženy v sekci 5.1.4. Aby však byla dodržena bezschématová podstata NoSQL databází, je třídou `ODMHandler` nabízena také metoda `get_db()`, jež vrací objekt driveru `PyMongo` dovolující v případě potřeby upravovat uložené dokumenty na nižší úrovni bez omezení v podobě nutnosti držet se předem definovaných atributů `DatabaseBackup`.

Backendová logika

Zálohování databází přineslo do backendového modulu `Database` další dva koncové body. První z nich je určen k manuálnímu spouštění vytváření nových záloh a k načítání seznamu dostupných záloh konkrétní databáze. Samotné soubory záloh jím kvůli velikosti přenášeny nejsou, pouze patřičná metadata (identifikátor, datum vzniku a důvod vzniku). Druhý koncový bod již zajišťuje stažení vybraného souboru zálohy. Těmto koncovým bodům není potřeba v požadavku posílat žádná data, identifikace cílové databáze je obstarána jejím identifikačním číslem v URL, rovněž tak identifikace konkrétní zálohy ke stažení.

Kvůli vytváření záloh byla v balíčku `api.admin.services` implementována metoda `backup_db()`. Jedním z jejích nepovinných parametrů je jméno souboru obsahujícího zálohu. Pokud není vyplněno, je vytvořen dočasný soubor, do nějž je následným voláním výše popsané metody `export_db()` vyexportována daná databáze – určená číselným identifikátorem předaným v povinném parametru – v podobě skriptu jazyka SQL. Tento způsob zálohování využívá všechny představené možnosti exportu. Výsledný skript tudíž obsahuje jak tabulky, tak všechna data. Soubor zálohy je pak v binární podobě, společně s identifikátorem databáze a pomocnými metadaty, uložen do databáze MongoDB.

Zálohovací metoda není volána vždy pouze na popud požadavku od uživatele. Některé akce ji volají automaticky, pokud je zálohovací modul zapnutý. K automatickým zálohám dochází například před smazáním databáze v uživatelském rozhraní nebo při importování uživatelských skriptů jazyka SQL (importy jsou popsány v následující sekci).

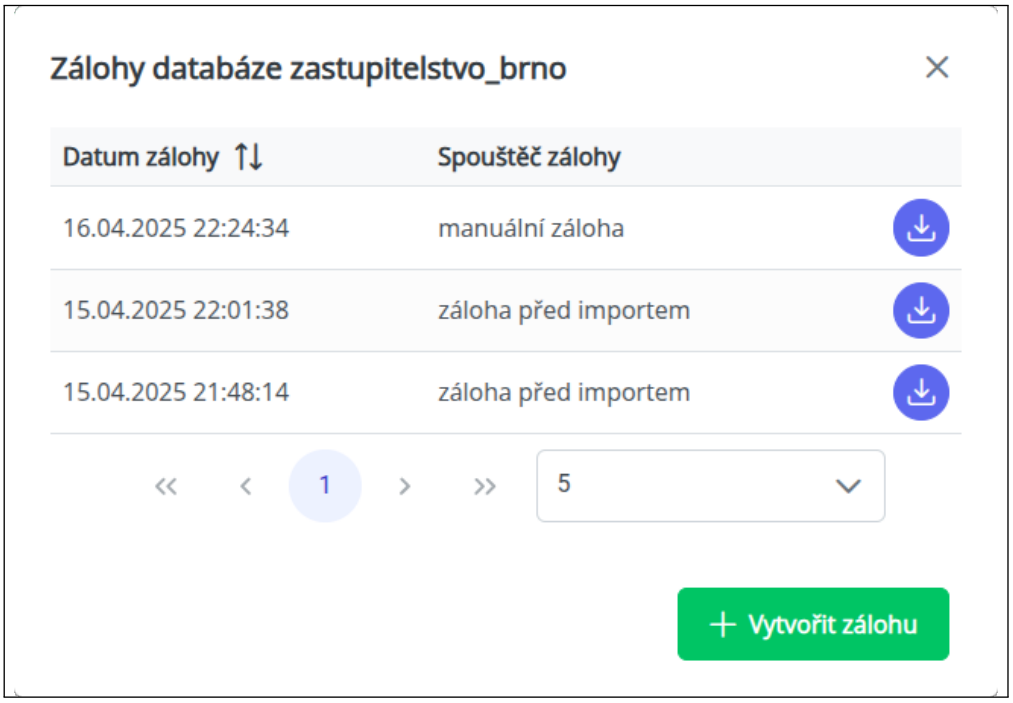
Uživatelské rozhraní

V klientské aplikaci byl přístup k zálohám databází přidán ke každé databázi v seznamu v podobě akčního tlačítka s ikonou záložky. Je pravděpodobně, že s postupem času budou databáze disponovat desítkami záloh, proto by mohlo být drahé při načtení seznamu databází server obratem žádat také o seznam záloh každé z nich. Frontend požadavek na server tedy odesílá až po kliknutí na akční tlačítko a uživateli je v mezičase zobrazen indikátor načítání.

Po načtení seznamu záloh nedochází k přesměrování na novou stránku, ale je otevřeno plovoucí okno komponenty `BackupDialog` viditelné na obrázku 6.2, které uživateli nabídne všechny historické zálohy zvolené databáze a možnost jakoukoliv stáhnout, respektive vytvořit novou. Identifikátor zálohy není uživateli zobrazován, protože jde o dlouhé hexadecimální

[4] https://mongoengine.org/

[5] Oficiální driver pro práci s MongoDB v jazyce Python, dokumentace je dostupná na https://www.mongodb.com/docs/languages/python/pymongo-driver/current/.



Obrázek 6.2: Plovoucí okno zobrazující seznam dostupných záloh vybrané databáze včetně relevantních metadat. Okno nabízí možnost stažení souboru zálohy nebo vytvoření nové zálohy aktuálního stavu databáze.

číslo typu `ObjectId`[6] a časové razítko k identifikaci člověkem v praxi stačí. Identifikátor je však používán interně a je automaticky vložen do URL při odeslání požadavku ke stažení.

6.1.5 Importování databází

Importování databází probíhá nahráním skriptu v jazyce SQL od uživatele. Aby metadata o databázích korektně zohledňovala všechny úpravy provedené nahranými skripty, není možné je pouze bez čtení spustit. Bylo proto nutné nalézt knihovnu, která pomůže se zpracováním textu skriptu do strojově jednodušeji čitelné podoby. Otestovány byly knihovny pro jazyk Python `sqlparse`, `sqlglot` a `mo-sql-parsing`. Poslední jmenovaná byla z testování vyřazena, protože se neuměla vypořádat s některými specifiky používaného SQL dialektu – například s existenciální klauzulí u příkazu `CREATE DATABASE` –, které `mariadb-dump` používaný pro export databází do skriptů vkládá. Pro zbývající dvě knihovny byly provedeny výkonnostní testy uvedené v tabulce 6.1, při nichž byly zpracovávány různě velké SQL soubory. Z těchto testů vyšla lépe knihovna `sqlglot`, která byla na testovaných datech velikostně odpovídajících aktuální realitě o necelou polovinu rychlejší.

Díky své rychlosti byla knihovna `sqlglot` vybrána pro použití v interní backendové metodě `import_sql()`, jež je volána jednoduchým koncovým bodem implementovaným v modulu `Database` pro přijímání SQL souborů v uživatelském požadavku. Dle návrhu v sekci 5.2.2 byl algoritmus používaný touto metodou rozdělen na dvě hlavní fáze. První fázi přibližuje obrázek 6.3. Během ní metoda iterativně prochází SQL příkazy parsované

[6] Výchozí identifikátor záznamů v databázích MongoDB. Více na https://mongodb.com/docs/manual/reference/bson-types/#std-label-objectid.

Velikost SQL souboru	`sqlparse`	`sqlglot`
500 kB	11,49 s	6,38 s
8 MB	201,93 s	104,58 s
24 MB	820,62 s	443,35 s

Tabulka 6.1: Tabulka zobrazuje čas potřebný pro dané knihovny k analyzování skriptu v jazyce SQL o uvedené velikosti. Velikost 500 kB odpovídá exportům menších databází aktuálně používaných v aplikaci. Přibližně 8 MB má v současnosti export největší databáze. Skript o velikosti 24 MB získán spojením exportů většího počtu databází. Probíhalo pět běhů měření na osobním stroji, z nichž byla na konci vypočítána průměrná hodnota.

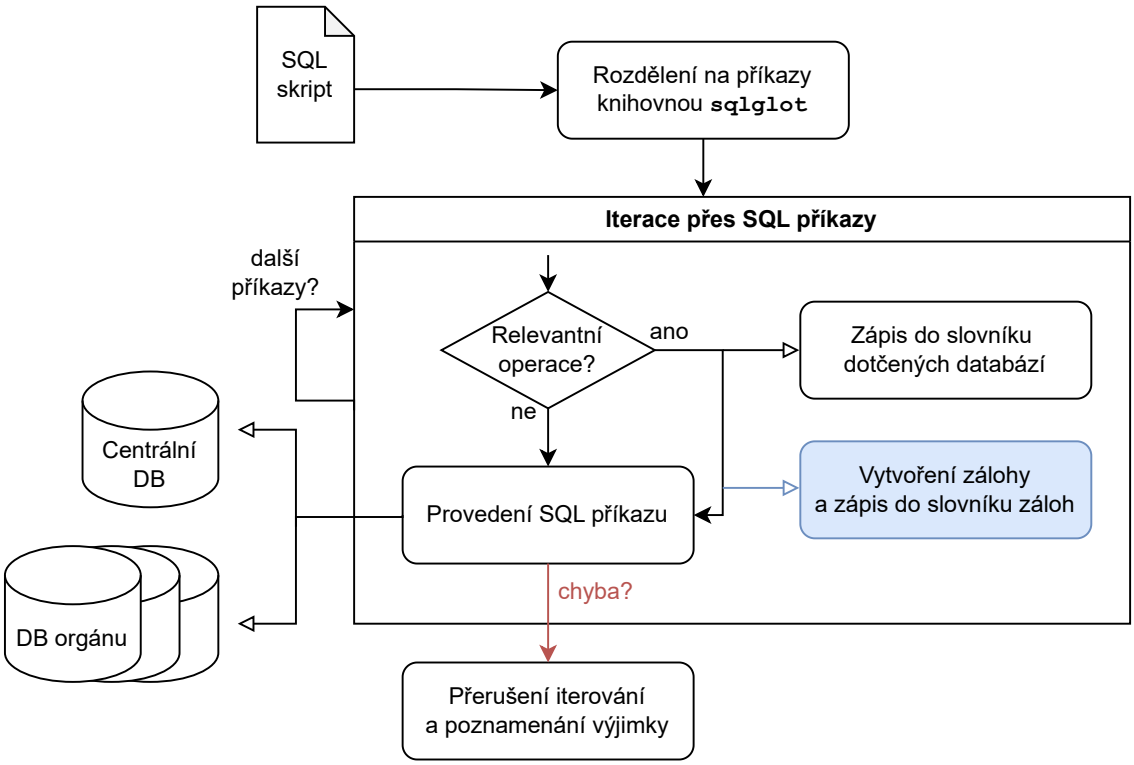
knihovnou `sqlglot` a hledá v nich relevantní operace, které byly při návrhu identifikovány jako `CREATE DATABASE`, `DROP DATABASE` a `USE`. Při nalezení je operace přidána do slovníku `affected_databases`, kde klíčem je jméno dotčené databáze a hodnotou seznam akcí v pořadí průběhu. V tomto bodě jsou také prováděny kontroly existenčních klauzulí. Příkazy typu `DROP DATABASE IF EXISTS <db_name>`, kde `<db_name>` je jméno neexistující databáze, nesmí způsobit zápis do slovníku dotčených databází, jelikož nemají v praktickém důsledku žádný efekt. Pokud je v aplikace povoleno zálohování, před všemi identifikovanými relevantními operacemi kromě `CREATE DATABASE` je zapsán export do dočasného souboru a odkaz na něj umístěn jako hodnota do slovníku `database_dumps`, kde klíčem je opět jméno databáze, pokud již záloha ve slovníku neexistuje. Na závěr zpracování každé iterace je SQL příkaz vykonán knihovnou `SQLAlchemy`. Pokud je při provádění příkazu vyvolána výjimka, je interně zaznamenána a iterování je přerušeno.

Po nečekaném i běžném dokončení iterování je započata druhá fáze algoritmu naznačená na obrázku 6.4. Ta v souladu s návrhem musí nastat i při výjimce v předchozí fázi, protože případné smazání nebo vytvoření nových databází provedené před výjimkou nelze jednoduše vrátit zpět. Metadata o databázích, tj. časová razítka jejich poslední změny a zálohy či příznak smazání, musí být tedy vždy aktualizována.

Druhá fáze algoritmu spustí iteraci přes slovník dotčených databází, kde prochází seznam uložených operací, které byly nad jednotlivými databázemi provedeny. Podle poslední vykonané operace a existence či neexistence nově vytvořeného exportu jsou aktualizována metadata v centrální databázi – včetně přidání záznamů o případných zcela nových databázích – a exporty jsou zálohovány na server s databází MongoDB, pokud je tato funkcionalita povolena.

Uživatelské rozhraní

Podle původního návrhu měl být prostor k nahrání skriptu v uživatelském rozhraní umístěn na stejné stránce spolu s možností tvorby prázdné databáze pomocí formuláře. V průběhu implementace bylo od tohoto nápadu upuštěno, protože nahrané skripty nemusí sloužit pouze k importu nových databází, ale také k aktualizacím existujících. Proto podobně jako pro ostatní prvky databázového modulu byl tento prostor výsledně implementován ve vlastním plovoucím okně v komponentě `ImportDialog`, která je používána jako součást větší komponenty `Databases`. Toto plovoucí okno obsahuje pouze jednoduchý formulář, který využívá `FileUpload` z knihovny `PrimeReact` jako komponentu pro výběr souboru. Po odeslání souboru je jeho zpracovávání serverem signalizováno indikátorem načítání.

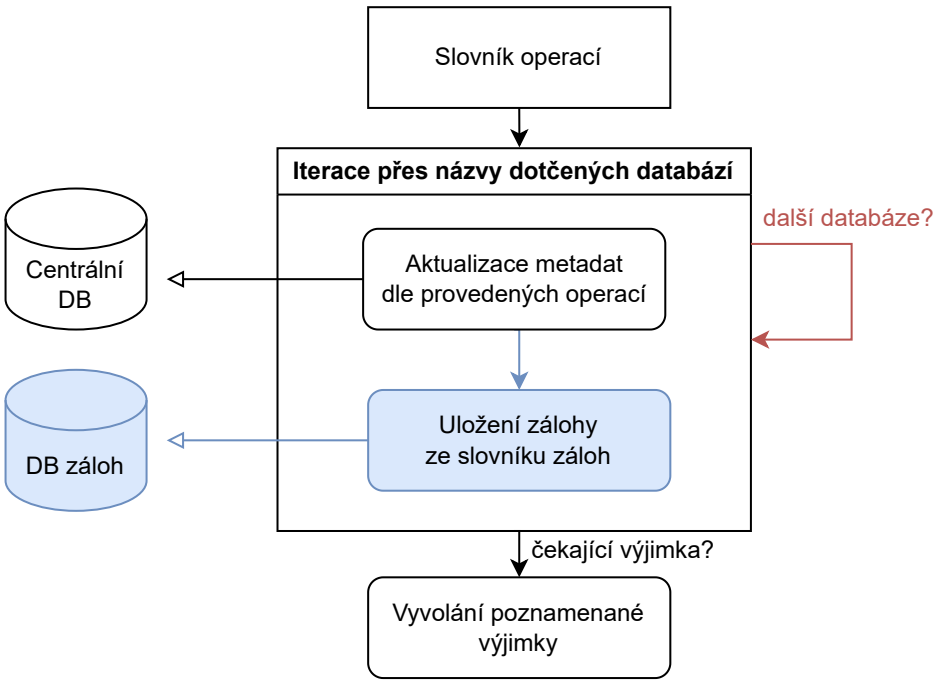


Obrázek 6.3: První fáze algoritmu pro import uživatelských SQL skriptů, které zobrazuje proces průchodu skriptem a zaznamenávání spouštěných operací. Za relevantní operace jsou považovány příkazy pro mazání, vytváření a selekce používané databáze pomocí `USE`. Algoritmus přechází do druhé fáze na konci iterování nebo při zachycené výjimce. Modrá barva značí akce prováděné pouze, když je připojena databáze MongoDB. Červený přechod je prioritní.

6.2 Generalizace správy politických orgánů

V návaznosti na požadavek definovaný sekcí 4.6.1 bylo nutné administrační modul modifikovat tak, aby umožňoval přidávat, upravovat i odstraňovat libovolné podporované typy orgánů místo stávajícího pevného navázání municipalit na svá zastupitelstva. Kvůli těmto změnám byly prováděny především úpravy frontendové části aplikace popsané dále. Implementace si však vyžádala také určité zásahy do backendu:

- **Úpravy přenášených datových souborů**
  Do schémat datových souborů přenášených spolu s HTTP požadavky bylo nutno zavést informaci o typu orgánu tam, kde původně systém implicitně počítal pouze se zastupitelstvem. Nově se přenáší typ požadovaného orgánu k vytvoření například při zakládání nové municipality nebo při uživatelské žádosti o založení nové municipality.
- **Promítnutí orgánů do odpovědí z API**
  Aplikační rozhraní při zasílání informací o municipalitách nyní spolu s každou municipalitou vrací také seznam všech přidružených orgánů.



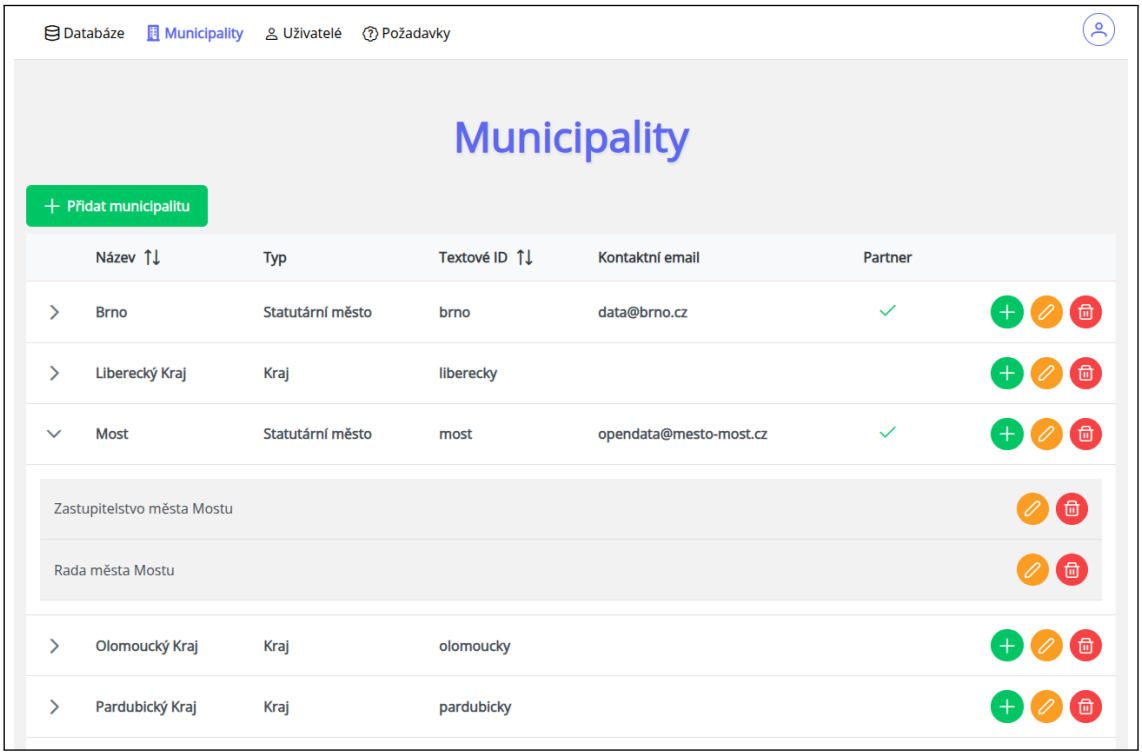
Obrázek 6.4: Druhá fáze algoritmu pro import uživatelských SQL skriptů, které ukazuje proces průchodu všech operací zaznamenaných během provádění skriptu a aktualizaci metadat o databázích tak, aby odpovídala finálnímu stavu. Algoritmus je ukončen, jakmile nemá přes co iterovat (v případě čekající výjimky až po jejím vyvolání). Modrá barva značí akce prováděné pouze, když je připojena databáze MongoDB. Červený přechod je prioritní.

- **Samostatná správa orgánů**
  V balíčku `api.admin` byly implementovány nové třídy nesoucí názvy `BodyTypeList`, `BodyList` a `BodyOne` k obsluze první trojice koncových bodů navržených v sekci 5.2.1, které dovolují samostatně vytvářet, upravovat a mazat politické orgány pod jednotlivými municipalitami. Endpointy pro operace vytváření a mazání byly omezeny pouze pro globální administrátory. Úpravy atributů orgánů však byly povoleny i běžným správcům municipalit.

6.2.1 Úpravy frontendu

Původní prefix URL adresy používaný k zobrazení všech pohledů souvisejících s úpravami funkčních období, zasedání a dalších podružných entit parametrizoval pouze municipalitu. Prefix neobsahoval výběr orgánu a uvažoval o zastupitelstvu jako o jediné možnosti. Byl proto rozšířen na `/admin/municipality/<id_municipality>/<body_type>`, kde řetězec `<body_type>` určuje textový identifikátor typu orgánu (v současné době *zastupitelstvo* nebo *rada*). Každá municipalita může disponovat jedním orgánem daného typu. Tento způsob rozlišení je tedy dostatečně unikátní a také konzistentní s řešením ve veřejné části aplikace, kde rozlišování orgánů bylo implementováno již dříve jako součást práce [30].

Výběr orgánů v administračním uživatelském rozhraní byl implementován dle návrhu v sekci 5.3 a výsledek je k vidění na obrázku 6.5. K jeho dosažení byla modifikována komponenta `MunicipalityTable`, která nyní nové informace o dostupných orgánech ob-



Obrázek 6.5: Snímek obrazovky ukazující upravený seznam municipalit, který nyní po expanzi jednotlivých řádků nabídne orgány dostupné pro danou municipalitu. Jedná se o pohled z účtu globálního administrátora, který navíc oproti běžným správcům může nové orgány přidávat a odebírat.

držené z API načítá do vnořené tabulky využívající komponenty `DataTable` z knihovny `PrimeReact`. Tato tabulka je zobrazena po expanzi řádku municipality kliknutím. Pro vytvoření nového orgánu pod zvolenou municipalitou byla implementována komponenta `BodyDialog`, která v plovoucím okně otevře nový formulář. Pokud jsou této komponentě předány data vybraného orgánu, její funkcionalita se přizpůsobí z vytváření nových orgánů na editaci existujícího. Vymazání orgánu je potvrzováno rovněž přes plovoucí okno implementované v komponentě `ConfirmDeleteBodyDialog`. Tlačítka pro přidávání a mazání orgánů jsou vykreslována pouze globálním administrátorům.

6.3 Sledování historie úprav

Změny datového modelu korespondující s historií úprav byly pevně ustáleny již během návrhu v kapitole 5, a to včetně popisu struktury tabulek ukládajících záznamy změn. Implementace se od návrhu v oblasti datového modelu téměř neodchýlila.

Schéma centrální databáze bylo doplněno o entity **VolitelnáVlastnost** a **MunicipalitaMáVlastnost** dovolující zaznamenat, jaké rozšiřující vlastnosti jsou aktivovány pro jednotlivé municipality. Sledování historie v této době zůstává jedinou rozšiřující vlastností. Řešení je však připraveno pro potenciální budoucí rozvoj. Schéma orgánových databází bylo doplněno o entity **HlasováníHistorie** a **HlasováníČlenaHistorie**. Oproti návrhu nebyla přidána entita **ZasedáníHistorie**, jelikož během týmových porad nebylo shledáno sledování historie úprav atributů jednotlivých zasedání jako dostatečně významné. Změny

existujících databází byly provedeny migračním skriptem v jazyce Python a inicializační skripty pro prázdné databáze byly patřičně upraveny tak, aby odpovídaly novému stavu.

Backendová logika

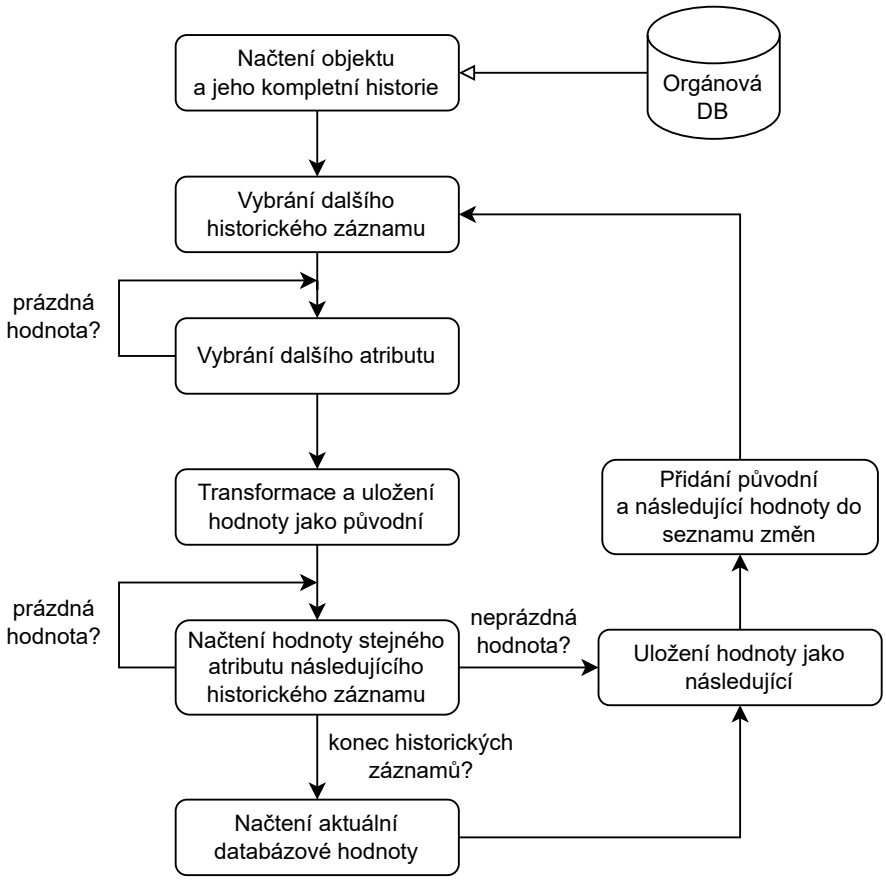
Koncový bod obsluhovaný třídou `VoteOne` musel být upraven tak, aby ve svých odpovědích posílal také historickou posloupnost úprav nejen pro atributy samotného hlasování, ale také pro jednotlivé hlasovací možnosti členů orgánu. Pokud pro dané hlasování existují historické záznamy, do navracené struktury ve formátu JSON je nyní přidáván seznam `edit_history` formátovaný podle návrhu v sekci 5.2.3. Tento formát je použitelný pro přenášení historie libovolné databázové entity a je sestavován pomocnou funkcí `build_edit_history()` dostupnou v celém administrátorském modulu z balíčku `api.admin.util.history_tracking`.

Při implementaci této funkce bylo dbáno na možnost použití s jakoukoliv databázovou entitou, u níž jsou nebo v budoucnu budou sledovány úpravy. V parametrech při volání je předáván seznam názvů sledovaných atributů schématu patřičného ORM modelu, což umožňuje potřebnou univerzalitu. Funkci je také možné předat transformační mapu v podobě slovníku, kde klíč je název atributu a hodnota volatelný objekt (typ `Callable` – obvykle funkce), jenž je v průběhu sestavování odpovědi volán nad hodnotami daného atributu. Toho je v případě historie hlasování využíváno například u atributů s typem `Time` knihovny `SQLAlchemy`, který není do JSONu automaticky serializovatelný a musí být nejprve transformován funkcí `format_time()`.

Celý algoritmus implementovaný ve funkci `build_edit_history()` je zjednodušeně popsán na obrázku 6.6. Jeho cílem je sestavit z historických záznamů časovou osu, kde každé časové razítko obsahuje původní hodnotu v daný moment změněných atributů a následující hodnotu, na niž byla původní změněna. Funkci je předán aktuální databázový objekt a jeho kompletní historie úprav. Funkce prochází historii od nejstaršího záznamu, kde hledá první neprázdný atribut, jehož hodnotu po případné transformaci uloží jako původní. Dále postupně prochází následující záznamy, v nichž hledá na co byla původní hodnota změněna. Pokud tuto hodnotu nenajde v historii, znamená to, že daný atribut doposud nebyl znovu upraven a hodnota se stále nachází v předaném databázovém objektu. Podobným způsobem je sestavena kompletní časová osa změn všech atributů.

Uživatelské rozhraní

Zobrazování historie bylo na frontendu implementováno úpravou komponent `VoteEdit` a do ní vkládané `VoteForm`, které vykreslují formulář na úpravu údajů konkrétního hlasování a zvolených hlasovacích možností členů orgánu. U hlasování, jež byla v minulosti upravena, je nyní zobrazována malá ikona knížky, která po rozkliknutí zobrazí plovoucí okno implementované komponentou `EditHistory` viditelné na obrázku 6.7. Stejná komponenta je využívána pro zobrazování úprav hodnoty hlasovací možnosti i pro úpravy atributů samotného hlasování jako je jeho předmět nebo čas konání. V jeden moment však okno vždy ukazuje úpravy pouze jediné vybrané entity.



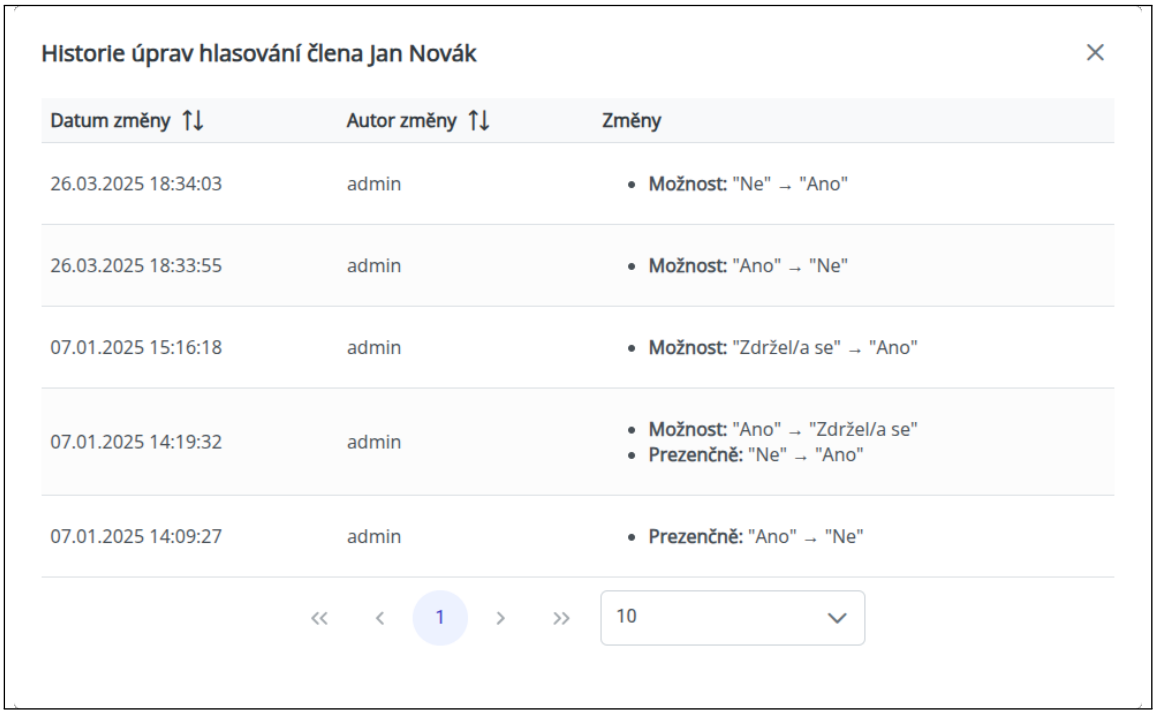
Obrázek 6.6: Zjednodušená ukázka algoritmu používaného ve funkci `build_edit_history_()` k sestavení časové osy úprav. Algoritmus prochází historické záznamy a pro každé časové razítko hledá původní hodnotu atributu a následující, na kterou byl atribut v daný moment změněn. Algoritmus končí, jakmile první cyklus projde všechny historické záznamy.

6.4 Nová knihovna aplikačního rozhraní

Aplikační rozhraní bylo dříve implementováno pomocí knihovny *Flask RESTful*[7], která však již nedostává pravidelné aktualizace a neobsahuje nativní podporu generování specifikace OpenAPI. Jako nová API knihovna byla vybrána autorem paralelně psané práce [29] knihovna *Flask Smorest*[8], která je v současnosti stále aktivně vyvíjena a oproti dříve používané podporuje nejen automatickou specifikaci OpenAPI, ale také generování dokumentace celého aplikačního rozhraní, a tedy vidinu snadnější spolupráce při jeho vývoji nebo při procesu hledání chyb.

Tato práce implementovala nahrazení staré knihovny za nově vybranou v rámci administračního modulu. Implementace v zájmu zachování konzistence proběhla podle vzoru navrženého pro práci [29] jejím autorem k využití ve veřejném modulu. Nejprve byly definovány třídy implementující abstraktní třídu `Blueprint` knihovny `Smorest` udávající URL prefix pro každý zdroj v administračním modulu (význam zdrojů dle paradigmatu REST

[7] https://flask-restful.readthedocs.io/en/latest/
[8] https://flask-smorest.readthedocs.io/en/latest/



Obrázek 6.7: Snímek obrazovky plovoucího okna ukazujícího úpravy hodnoty hlasovací možnosti člena provedené z administračního modulu. Pořadí lze filtrovat podle data nebo autora změny. Pokud bylo upraveno více možností v jeden moment, jsou zobrazeny dohromady. Tato situace je k vidění u předposledního zobrazeného řádku.

byl popsán v sekci 3.3.1). Třídy administračního modulu obsluhující koncové body byly pak za použití dekorátorů připojeny k těmto blueprintům tak, aby byly při směrování požadavků správně volány novou knihovnou.

Při procesu integrace knihovny byl navíc změněn implicitní stavový kód protokolu HTTP, který server vrací při sémanticky nesprávném, avšak syntakticky korektním požadavku, z `400 Bad Request` na `422 Unprocessable Content`, který dle standardu tuto situaci vystihuje lépe[9].

Před úpravami bylo kvůli navracení chybových odpovědí klientovi ve vlastním formátu využíváno metody `handle_error()` třídy `ExtendedApi` implementované spolu s diplomovou prací [30]. Tato metoda přepisovala implicitní chování původní API knihovny při vzniknuvších výjimkách. Řešení však po nahrazení knihovny nefungovalo, jelikož *Smorest* aplikuje jiný přístup k ošetřování chyb, jemuž bylo třeba se přizpůsobit. Metody ošetřující výjimky jsou proto nově registrovány na úrovni jednotlivých blueprintů aplikačního rozhraní pomocí vestavěného dekorátoru `errorhandler`. Z důvodu zachování konzistence mezi jednotlivými moduly i nové ošetřování chyb bylo navrženo ve spolupráci s autorem práce [29], v rámci níž byly podobné zásahy provedeny ve veřejném modulu.

[9]Popisy obou stavových kódu jsou dostupné na https://rfc-editor.org/rfc/rfc9110.html#name-client-error-4xx.

Kapitola 7

Testování

Tato kapitola popisuje testování výsledné implementace, které probíhalo průběžně při každém dokončení většího milníku zasahujícího do aplikačního nebo uživatelského rozhraní či do databázových schémat. Testováno bylo dvěma způsoby představenými v následujících sekcích.

7.1 Automatizované testování

Automatizované testy byly používány pro testování aplikačního rozhraní. Využíváno bylo testovacího prostředí vytvořeného k práci [30] postaveného na frameworku `pytest`. V prostředí je používán testovací klient k odesílání HTTP požadavků na aplikační rozhraní. Stavové kódy a případná data poslaná v odpovědích jsou následně porovnávány s očekávanými výsledky. Testovací framework plní předem zadané rutiny definované v testovacích modulech, kde každý je typicky určen k testování funkcionality jedné databázové entity. Kromě drobnějších úprav ve stávajících modulech byly v této práci přidány dva nové.

7.1.1 Testování rozhraní politických orgánů

Modul byl implementován v souboru `test_body.py` a ověřuje správnost aplikačního rozhraní k samostatné správě politických orgánů. Tento modul obsahuje celkově 18 testů, které prověřují funkčnost všech koncových bodů implementovaných v rámci generalizace orgánů. Kontrolovány jsou následující vlastnosti:

- načtení typů orgánů,
- načtení informací o orgánu,
- vytvoření orgánu,
- úprava orgánu,
- smazání orgánu,
- funkčnost uživatelských oprávnění.

Prověřovány nejsou jen úspěšné průběhy, ale také situace vracející chybové stavové kódy. Obecně jde o testy neautorizovaných čtení či dalších operací a kontroly chybějících polí v datech odesílaného požadavku nebo polí porušujících nastavená omezení jako například maximální délka řetězce.

Jeden z testů specifických pro tento modul kontroluje, zda je po odstranění orgánu možné pro municipalitu znovu vytvořit orgán stejného typu. Test byl implementován na popud manuální kontroly, při níž došlo k uvědomění, že po smazaném orgánu zůstává databáze, která kvůli konvencím, podle nichž jsou orgánové databáze pojmenovávány (vysvět-

leno v sekci 4.4.2), způsobovala jmenné konflikty při opětovném založení orgánu stejného typu pod danou municipalitou. Tato situace vedla na automatický pokus o přidání stejně pojmenované databáze. Z důvodu prevence omylů nebylo přijatelné, aby smazání orgánu nenávratně odstranilo také celou jeho databázi. Tento problém byl vyřešen samovolným přidáním přípony s časovým razítkem smazání za jméno databáze ve chvíli, kdy je k ní přiřazený orgán odstraněn.

7.1.2 Testování rozhraní databázového modulu

Modul testující databázové operace se nachází v souboru `test_database.py` a prověřuje koncové body implementované jako součást nového modulu ke správě databází. Obsahuje celkově 26 automatizovaných testů, které sledují správnou funkčnost těchto vlastností:

- načtení typů orgánů,
- načtení jedné či více databází,
- vytvoření databáze,
- přejmenování databáze,
- export databáze,
- záloha databáze,
- smazání databáze,
- import skriptu jazyka SQL.

Podobně jako předchozí testovací modul také tento prověřuje úspěšné i neúspěšné průběhy a z obecných vlastností jsou testovány neautorizované operace nebo chybné formáty odesílaných požadavků. Některé z testů interagují s databází MongoDB, která je však nepovinnou složkou systému (více informací viz sekce 6.1.4). Pro tyto testy bylo vytvořeno nové označení (tzv. *marker*) pro knihovnu `pytest`[1]. Konfigurace testů v souboru `conftest.py` byla upravena tak, aby takto označené testy byly přeskočeny, pokud v prostředí není definována proměnná `USE_MONGO` na hodnotu `1`. Připojovací údaje k testovací databázi MongoDB se taktéž definují v prostředí pomocí proměnných odlišených od těch, co jsou používány ke konfiguraci připojení k databázi pro ostrý provoz.

Testy, které ověřují import skriptů v jazyce SQL, používají předem připravený skript `empty-test-db.sql`. Testovací databáze by však měly respektovat možnost uživatele nastavit si prefix předcházející jejich názvy pomocí proměnné prostředí, která byla zavedena prací [30], což by nebylo splněno při obyčejném spuštění skriptu obsahujícího staticky určené jméno databáze. Připravený skript proto má ve všech příkazech, kde je jméno databáze použito, řetězec `{{custom_name}}`. K přečtení tohoto skriptu testy používají pomocnou funkci `load_sql_with_custom_db_name()`, která řetězec nahradí reálným jménem databáze, jenž jí je předáno parametrem. Tímto způsobem testy mohou vyhovět proměnlivým prefixům názvů testovacích databází i při vytvoření databáze ze schématu obsaženém ve statickém skriptu.

7.2 Uživatelské testování

Z hlediska uživatelského rozhraní tato práce nejvíce zasáhla do části, jež je přístupná pouze globálním administrátorům, tzn. typicky členům týmu projektu, u nichž je předpokládána vysoká úroveň technických znalostí. Všechny změny klientské aplikace byly průběžně nasazovány do vývojové větve projektu společně s odpovídajícími úpravami aplikačního rozhraní,

[1] Více o práci s markery na https://docs.pytest.org/en/stable/example/markers.html.

kde byly týmu demonstrovány a připraveny k vlastnoručnímu vyzkoušení. To pomohlo odhalit některé chyby nedopatřením zavedené do frontendové aplikace, která není pokryta automatizovanými testy. Typicky šlo o chyby vyplývající z přístupů k nedefinovaným objektům související s nesynchronními úpravami stavů knihovny `React`.

Po vyzkoušení rozhraní členy týmu, kteří rozhraní databázového modulu budou používat nejvíce, byly na základě zpětné vazby upraveny také aspekty některých ovládacích prvků. Mezi těmito změnami bylo přidání zaškrtávacího pole k seznamu databází, jehož aktivace označí k exportu všechny databáze pomocí jednoho kliknutí. Tlačítko ke smazání databáze se v určité fázi implementace nacházelo vedle ostatních akčních tlačítek na řádku každé databáze, ale jako reakce na zpětnou vazbu bylo přesunuto do formuláře úprav. K potvrzení odstranění je nyní navíc třeba opsat jméno databáze, aby byla minimalizována šance omylů a nenávratné ztráty dat, pokud by aplikace nebyla zároveň připojena k zálohovací databázi a nemohlo dojít k automatické záloze před smazáním.

Kapitola 8

Závěr

Cílem této diplomové práce bylo rozšířit administrační rozhraní projektu Zastupko.cz o možnost spravovat, zálohovat a importovat databáze. Jedním z důležitých vedlejších cílů pak byla generalizace interní správy samosprávných orgánů municipalit.

Teoretická část práce nabídla základní přehled politických orgánů České republiky na úrovni obcí i krajů včetně výtahu z významných částí patřičné legislativy. Popsán byl také způsob hlasování orgánů a přístup obcí a krajů České republiky k otevřenosti dat z těchto hlasování. Následně byly představeny základní koncepty a technologie oblasti informačních a redakčních systémů. Možnosti čtyř ze současně nejvýznamnějších redakčních systémů byly porovnány s důrazným zaměřením na jejich administrační rozhraní.

Dále byl analyzován současný stav projektu Zastupko.cz – jeho cíloví uživatelé, role v systému, použité technologie a datový model, ale také některé nedostatky jeho administračního modulu a obdržené požadavky na jeho vylepšení, podle nichž byl vytvořen návrh rozšíření datového modelu spolu s aktualizacemi aplikačního a uživatelského rozhraní.

Navržená rozšíření byla úspěšně implementována a v současnosti jsou nasazena nebo čekají na nasazení do vývojové větve projektu, z nichž by eventuálně měla být přesunuta do ostrého provozu. Nový přehled databází by interním administrátorům měl umožnit snadnější přístup k používaným databázím a jejich přenos mezi jednotlivými větvemi projektu. Zejména pak dovoluje vytváření databázových záloh a jejich ukládání do separátní databáze. Generalizace práce s orgány zase přináší všem správcům municipalit možnost upravovat data různých typů orgánů místo omezení na zastupitelstva, jak tomu bylo v dřívější verzi systému.

Navazující práce by mohly dále rozšířit možnosti exportu databází. Místo exportu celé dané databáze by systém mohl dávat uživateli na výběr stáhnout pouze určité logicky související celky jako například konkrétní funkční období a všechna jeho zasedání. Další možností by byly exporty na nižší úrovni dovolující výběr a stažení konkrétních databázových tabulek. Dále by funkcionalita zobrazující v administračním rozhraní historii manuálních úprav hlasování mohla být rozšířena na další databázové entity a historie by mohla být zobrazována také ve veřejném modulu pro zajištění větší transparentnosti zásahů správců do zobrazovaných dat.

Literatura

[1] Abdalhakim, H. Addressing Burdens of Open Database Connectivity Standards on the Users. In: Luo, Q. a Zhu, M., ed. *2009 Third International Symposium on Intelligent Information Technology Application Workshops*. IEEE Computer Society, 2009. ISBN 9781424464203. Dostupné z: https://doi.org/10.1109/IITAW.2009.40.

[2] Barker, D. *Web Content Management: Systems, Features, and Best Practices*. 1. vyd. O'Reilly Media, 2016. ISBN 9781491908129.

[3] Bergstein, A. *Drupal 10 Masterclass: Build responsive Drupal applications to deliver custom and extensible digital experiences to users*. 1. vyd. Packt Publishing, 2023. ISBN 9781837633104.

[4] Bernus, P.; Mertins, K. a Schmidt, G. *Handbook on Architectures of Information Systems*. 2. vyd. Springer Berlin Heidelberg, 2016. International Handbooks on Information Systems. ISBN 9783662517406.

[5] Berson, A. *Client/Server Architecture*. 2. vyd. McGraw Hill, 1996. ISBN 9780070056640.

[6] Boiko, B. *Content Management Bible (2nd Edition)*. 2. vyd. Wiley, 2004. ISBN 9780764573712.

[7] Buna, S. *GraphQL in Action*. 1. vyd. Manning, 2021. ISBN 9781617295683.

[8] Buňata, T. *Srovnání REST, GraphQL a gRPC API v Node.js*. Praha, CZ, 2022. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Dostupné z: https://dspace.cvut.cz/handle/10467/101057.

[9] Buytaert, D. *Drupal CMS 1.0 released* online. 2025. Dostupné z: https://dri.es/drupal-cms-1-released. [cit. 2025-01-21].

[10] Byron, L. *GraphQL: A data query language* online. 2015. Dostupné z: https://engineering.fb.com/2015/09/14/core-infra/graphql-a-data-query-language/. [cit. 2024-11-16].

[11] Česko. Zákon č. 491 ze dne 11. května 1999 o svobodném přístupu k informacím – znění od 1. července 2024. In: *Sbírka zákonů České republiky*. 1999, částka 39, s. 2578–2582. ISSN 1211-1244. Dostupné z: https://www.e-sbirka.cz/sb/1999/106/2024-07-01.

[12] Česko. Zákon č. 128 ze dne 12. dubna 2000 o obcích (obecní zřízení) – znění od 1. ledna 2025. In: *Sbírka zákonů České republiky*. 2000, částka 38, s. 1737–1764. ISSN 1211-1244. Dostupné z: https://www.e-sbirka.cz/sb/2000/128/2025-01-01.

[13] Česko. Zákon č. 129 ze dne 12. dubna 2000 o krajích (krajské zřízení) – znění od 1. ledna 2024. In: *Sbírka zákonů České republiky*. 2000, částka 38, s. 1765–1782. ISSN 1211-1244. Dostupné z: https://www.e-sbirka.cz/sb/2000/129/2024-01-01.

[14] Česko. Zákon č. 130 ze dne 12. dubna 2000 o volbách do zastupitelstev krajů a o změně některých zákonů – znění od 2. března 2024. In: *Sbírka zákonů České republiky*. 2000, částka 38, s. 1783–1799. ISSN 1211-1244. Dostupné z: https://www.e-sbirka.cz/sb/2000/130/2024-03-02.

[15] Česko. Zákon č. 131 ze dne 13. dubna 2000 o hlavním městě Praze – znění od 1. července 2024. In: *Sbírka zákonů České republiky*. 2000, částka 39, s. 1802–1834. ISSN 1211-1244. Dostupné z: https://www.e-sbirka.cz/sb/2000/131/2024-07-01.

[16] Česko. Zákon č. 491 ze dne 6. prosince 2001 o volbách do zastupitelstev obcí a o změně některých zákonů – znění od 2. srpna 2021. In: *Sbírka zákonů České republiky*. 2001, částka 178, s. 11002–11024. ISSN 1211-1244. Dostupné z: https://www.e-sbirka.cz/sb/2001/491/2021-08-02.

[17] Coulouris, G.; Dollimore, J.; Kindberg, T. a Blair, G. *Distributed Systems: Concepts and Design*. 5. vyd. Pearson, 2012. ISBN 9780132143011.

[18] Dolifka, D. *Conducting Experimental Research on Shopify: An Initial Discussion and Guide to Getting Started*. 2023. Dostupné z: https://doi.org/10.2139/ssrn.4451011.

[19] Drupalize.Me. *Drupal CMS User Guide* online. 2025. Dostupné z: https://new.drupal.org/docs/drupal-cms. [cit. 2025-01-21].

[20] Elmasri, R. a Navathe, S. *Fundamentals of Database Systems*. 7. vyd. Pearson, 2016. ISBN 9780133970777.

[21] Engel, D.; West, R.; Thomas, M.; Roth, J.; Guyer, C. et al. *ODBC Overview* online. 2023. Dostupné z: https://learn.microsoft.com/en-us/sql/odbc/reference/odbc-overview?view=sql-server-ver16. [cit. 2024-11-18].

[22] Fielding, R. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, California, USA, 2000. Disertační práce. University of California, Irvine. Dostupné z: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[23] Grinberg, M. *Flask Web Development: Developing Web Applications with Python*. 1. vyd. O'Reilly Media, 2014. ISBN 9781449372620.

[24] Hodgdon, J. et al. *Drupal User Guide* online. 2025. Dostupné z: https://www.drupal.org/docs/user_guide/en/index.html. [cit. 2025-01-21].

[25] Indrasiri, K. a Kuruppu, D. *GRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes*. 1. vyd. O'Reilly Media, 2020. ISBN 9781492058335.

[26] Janošík, A. *Nástroj pro zpracování dat z hlasování obecních zastupitelstev*. Brno, CZ, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/154549.

[27] Jin, B.; Sahni, S. a Shevat, A. *Designing Web APIs: Building APIs That Developers Love*. 1. vyd. O'Reilly Media, 2018. ISBN 9781492026921.

[28] Johansson, M. a Isabella, O. *Comparative Study of REST and gRPC for Microservices in Established Software Architectures*. 2023. Dostupné z: https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-195563.

[29] Kodaj, R. *Aplikační rozhraní systému hlasování zastupitelstev*. Brno, CZ, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/161352.

[30] Krejčí, O. *Správa municipalit v systému hlasování městských zastupitelstev*. Brno, CZ, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/154362.

[31] Kurose, J. a Ross, K. *Computer Networking: A Top-down Approach*. 7. vyd. Pearson, 2017. ISBN 9780133594140.

[32] Marakas, G. a O'Brien, J. *Introduction to Information Systems*. 16. vyd. McGraw Hill, 2013. ISBN 9780073376882.

[33] Meshen, L. *CMS Architecture: Traditional, decoupled & headless* online. 2021. Dostupné z: https://www.acrocommerce.com/article/cms-architecture-traditional-decoupled-headless. [cit. 2024-12-15].

[34] Messenlehner, B. a Coleman, J. *Building Web Apps with WordPress: WordPress as an Application Framework*. 2. vyd. O'Reilly Media, 2019. ISBN 9781491990087.

[35] Niswar, M.; Arisandy Safruddin, R.; Bustamin, A. a Aswad, I. Performance evaluation of microservices communication with REST, GraphQL, and gRPC. *International Journal of Electronics and Telecommunications*. 1. vyd. Polish Academy of Sciences Committee of Electronics and Telecommunications, 2024, sv. 70, č. 2. Dostupné z: https://doi.org/10.24425/ijet.2024.149562.

[36] Oracle. *Java JDBC API* online. 2024. Dostupné z: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/. [cit. 2024-11-18].

[37] Oracle. *What is a content management system (CMS)?* online. 2024. Dostupné z: https://www.oracle.com/cz/content-management/what-is-cms/. [cit. 2024-12-15].

[38] Pamukoff, W. *Shopify Functions: The New Way to Extend and Customize Shopify* online. 2022. Dostupné z: https://www.shopify.com/enterprise/blog/shopify-functions. [cit. 2025-01-02].

[39] Piccoli, G. a Pigni, F. *Information Systems for Managers: With Cases*. 4. vyd. Prospect Press, 2019. ISBN 9781943153503.

[40] Porcello, E. a Banks, A. *Learning GraphQL: Declarative Data Fetching for Modern Web Apps*. 1. vyd. O'Reilly Media, 2018. ISBN 9781492030713.

[41] Prasad, A. A.; Sharma, N. A. a Kumar, A. A Comparative Analysis of Joomla, Drupal, Wordpress and ASP.NET: Exploring Features, Performance, and Suitability. In: Institute of Electrical and Electronics Engineers, Inc. *2023 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. Los Alamitos, CA, USA: IEEE Computer Society, 2023. ISBN 9798350341089. Dostupné z: https://doi.org/10.1109/CSDE59766.2023.10487657.

[42] Publications Office of the European Union; Page, M.; Behrooz, A. a Moro, M. *Open Data Maturity Report 2024*. Publications Office of the European Union, 2024. Dostupné z: https://doi.org/10.2830/8656811.

[43] Rainer, R.; Prince, B. a Cegielski, C. *Introduction to Information Systems: Supporting and Transforming Business*. 5. vyd. Wiley, 2013. ISBN 9781118674369.

[44] Ratnayake, R. N. *Wordpress Web Application Development: Building robust web apps easily and efficiently*. 1. vyd. Packt Publishing, 2017. ISBN 9781787126800.

[45] Ratnayake, R. N. *WordPress 5 Cookbook*. 1. vyd. Packt Publishing, 2020. ISBN 9781838986506.

[46] Richardson, L.; Amundsen, M. a Ruby, S. *RESTful Web APIs: Services for a Changing World*. 1. vyd. O'Reilly Media, 2013. ISBN 9781449358068.

[47] Shopify. *Dokumentace systému Shopify* online. 2025. Dostupné z: https://shopify.dev/docs. [cit. 2025-01-02].

[48] Smith, A. *Database Drivers: Your Bridge to the Database* online. 2023. Dostupné z: https://medium.com/@averydcs/database-drivers-your-bridge-to-the-database-ad0d25d13d55. [cit. 2024-11-18].

[49] Snodgrass, E. a Soon, W. API practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange. *First Monday*. 1. vyd., 2019, sv. 24, č. 2. Dostupné z: https://doi.org/10.5210/fm.v24i2.9553.

[50] So, P. *Decoupled Drupal in Practice: Architect and Implement Decoupled Drupal Architectures Across the Stack*. 1. vyd. Apress, 2018. ISBN 9781484240717.

[51] Stair, R. a Reynolds, G. *Principles of Information Systems*. 9. vyd. Cengage Learning, 2010. ISBN 9780324665284.

[52] Turban, E.; Pollard, C. a Wood, G. *Information Technology for Management: On-Demand Strategies for Performance, Growth and Sustainability*. 11. vyd. Wiley, 2018. ISBN 9781118890790.

[53] W3Tech. *Market share yearly trends for content management systems* online. 2024. Dostupné z: https://w3techs.com/technologies/history_overview/content_management/ms/y. [cit. 2024-12-25].

[54] W3Tech. *Usage statistics and market shares of content management systems* online. 2024. Dostupné z: https://w3techs.com/technologies/overview/content_management. [cit. 2024-12-25].

[55] White, M. *The Content Management Handbook*. 1. vyd. Facet Publishing, 2005. ISBN 9781856045339.

[56] Wix. *Dokumentace systému Wix* online. 2025. Dostupné z: https://dev.wix.com/docs. [cit. 2025-01-03].

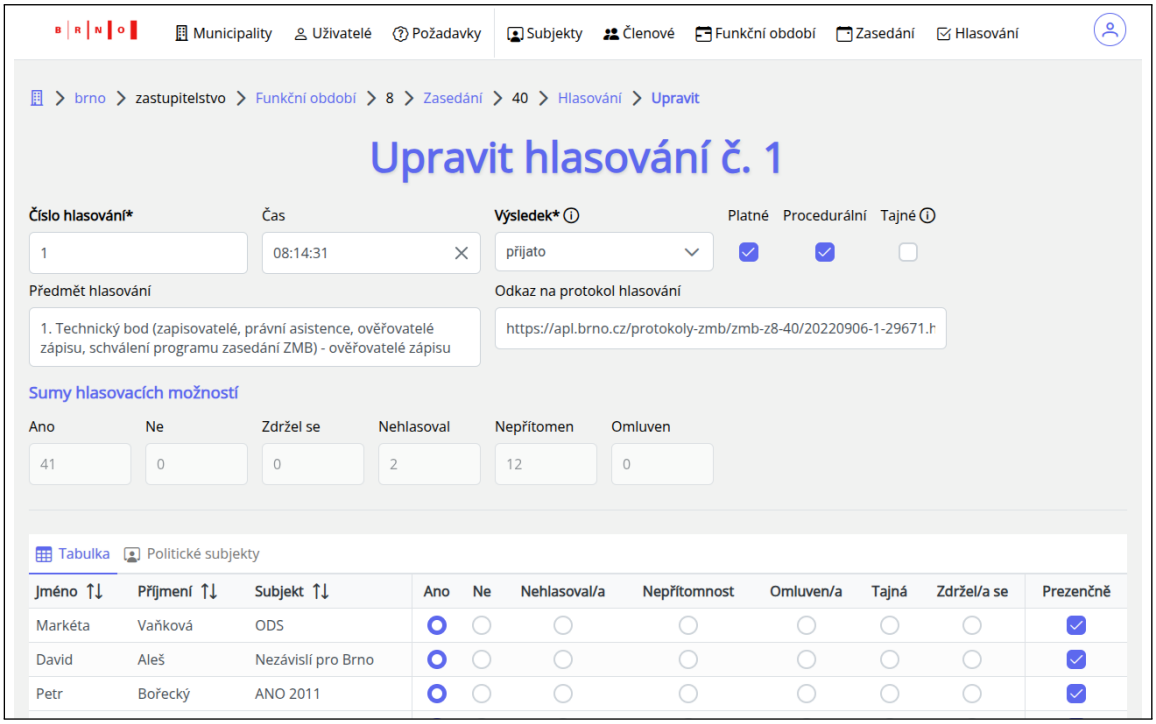
[57] Zaklová, K. *Analýza a vizualizace dat z hlasování Zastupitelstva města Brna*. Brno, CZ, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/146480.

[58] Zaklová, K. *Formal Models of Open Data*. Brno, CZ, 2025. Pojednání k tématu disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

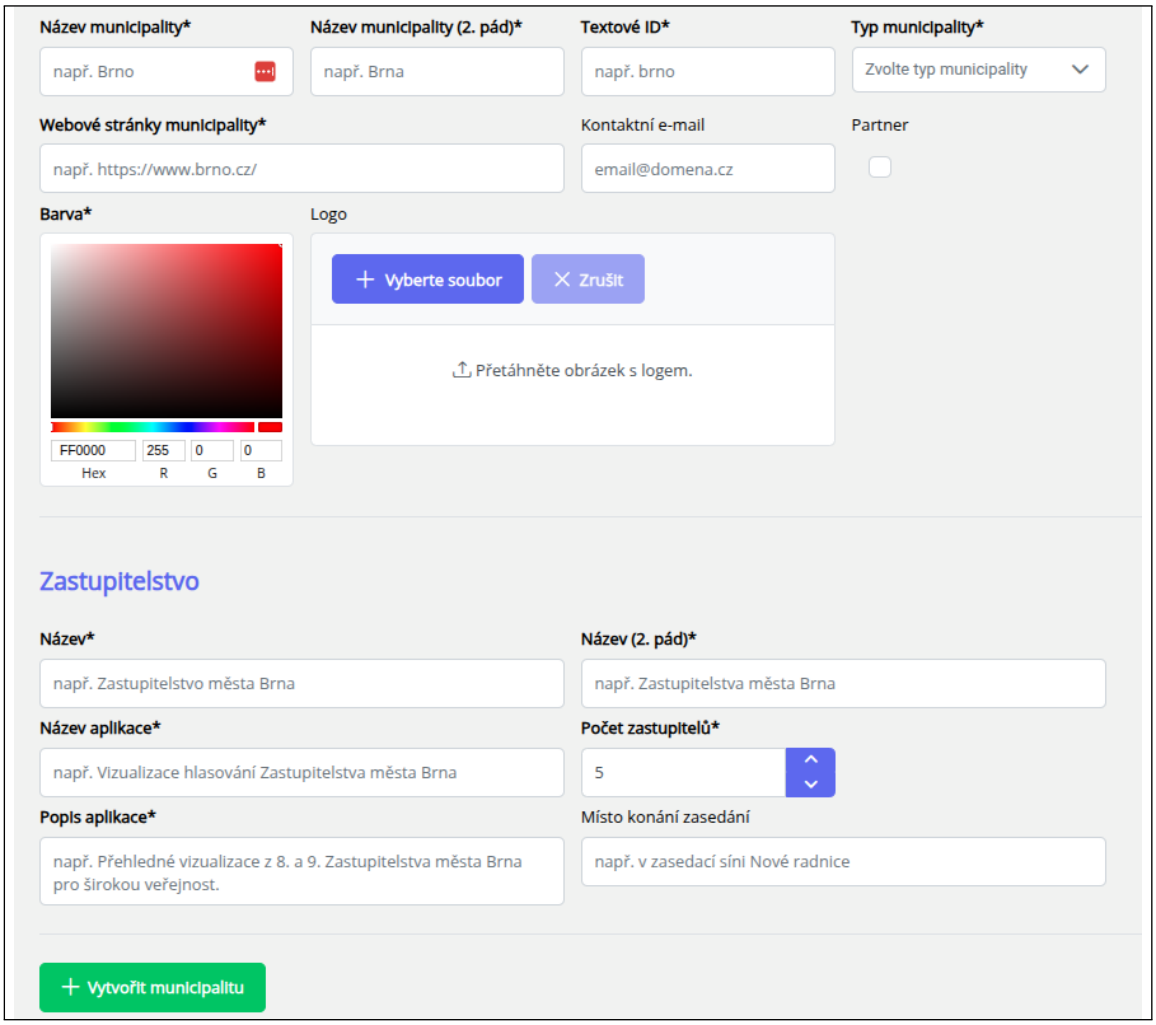
[59] Zbožínková Šárka. *Tvorba otevřených formálních norem: případová studie*. Brno, CZ, 2022. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Dostupné z: https://is.muni.cz/th/c84we/?studium=616837;id=400390.

Příloha A

Snímky obrazovky původního rozhraní projektu



Obrázek A.1: Snímek obrazovky ukazující rozhraní pro úpravu atributů hlasování a zvolených hlasovacích možností. Seznam členů byl pro potřeby snímku zkrácen.



Obrázek A.2: Snímek obrazovky ukazující formulář k vytvoření nové municipality a jejího zastupitelstva. Stejná komponenta je používána také pro úpravu existujících municipality a zastupitelstev. Okolní rozhraní bylo kvůli velikosti ořízнuto.