VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

APLIKAČNÍ ROZHRANÍ SYSTÉMU HLASOVÁNÍ ZASTUPITELSTEV

APPLICATION INTERFACE OF THE COUNCIL VOTING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. RADOSLAV KODAJ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. KRISTÝNA ZAKLOVÁ**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

161352

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Kodaj Radoslav, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Informační systémy a databáze
Název:	**Aplikační rozhraní systému 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 architektur serverových částí webových aplikací, aplikačních rozhraní a jejich specifikace.
3. Seznamte se se systémem hlasování zastupitelstev, a to především s jeho serverovou částí. Analyzujte současný stav a problémy.
4. Na základě zjištěných poznatků navrhněte novou architekturu serverové části systému a vyberte konkrétní nástroj pro vytvoření specifikace aplikačního rozhraní.
5. Po konzultaci s vedoucí navržené úpravy implementujte.
6. Vytvořené řešení otestujte a zhodnoťte dosažené výsledky.

Literatura:

- Richardson, L., Amundsen, M. *RESTful Web APIs*. Sebastopol: O'Reilly, 2013. ISBN 978-1-4493-5806-8.
- Martin, R. C. *Clean architecture: a craftsman's guide to software structure and design*. London, England: Prentice Hall, 2018. ISBN 978-0-13-449416-6.
- 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

Práca sa zaoberá vytvorením novej architektúry serverovej časti aplikácie pre zobrazovanie výsledkov hlasovania zastupiteľstiev v Českej republike. Cieľom je tiež integrovanie vybraného nástroja pre špecifikáciu aplikačného rozhrania a jej interaktívnej dokumentácie. Pre nástroj, ktorý vytvára špecifikáciu aplikačného rozhrania, bolo použité OpenAPI verzie 3.0. Vytvorené riešenie poskytuje funkčnú aplikačnú logiku, prehladnejšiu súborovú štruktúru a interaktívnu dokumentáciu API. Prínosom tejto práce je vytvorenie modulárnejšej štruktúry, ktorá uľahčuje ďalší vývoj a integrovanie nových časti do systému v budúcnosti. Taktiež vďaka novej dokumentácii OpenAPI sa zabezpečuje jednoduchšia testovateľnosť koncových bodov rozhrania. Výsledky tejto práce by mali uľahčiť ďalší následný vývoj.

Abstract

This work focuses on the development of a new server-side architecture for an application that displays the voting results of municipal councils in the Czech Republic. The aim is also to integrate a selected tool for API specification and its interactive documentation. For generating the API specification, OpenAPI version 3.0 was used. The resulting solution provides functional application logic, a clearer file structure, and interactive API documentation. The contribution of this work lies in creating a more modular structure, which facilitates further development and the integration of new components into the system in the future. Additionally, the new OpenAPI documentation ensures easier testability of API endpoints. The results of this work are intended to support and simplify future development efforts.

Kľúčové slová

Zastupiteľstvo, samospráva, aplikačné rozhranie, RESTful API, serverová časť aplikácie, koncový bod, špecifikácia API, architektúra

Keywords

Council, self-government, application interface, RESTful API, server part of application, endpoint, API specification, architecture

Citácia

KODAJ, Radoslav. *Aplikační rozhraní systému hlasování zastupitelstev*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Kristýna Zaklová

Aplikační rozhraní systému hlasování zastupitelstev

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod odborným vedením pani Ing. Kristýny Zaklovej. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Radoslav Kodaj
20. mája 2025

Poďakovanie

Rád by som poďakoval vedúcej práce Ing. Kristýne Zaklovej za veľmi odborné vedenie práce, poskytnutie rady a spätnej väzby. Ďalej chcem taktiež poďakovať Ing. Petrovi Johnovi za ústretovosť, pomoc pri nasadzovaní nových častí systému a poskytnuté konzultovanie implementačných detailov práce.

Obsah

Kapitola 1

Úvod

K blížiacim sa voľbám v Českej republike narastá prirodzene aj záujem zisťovať si informácie o jednotlivých kandidátoch, aby občan mohol vykonať čo najlepšie rozhodnutie pri plnení svojej občianskej povinnosti. V dnešnej dobe je v tomto ohľade množstvo informácií aj dezinformácií a je ťažšie sa vedieť zorientovať. Niekedy môže byť aj demotivujúce a náročné porovnávať rôzne analýzy z iných zdrojov. Je preto žiaduce, aby dáta o politických subjektoch a ich členoch boli centralizované a dostupné pre všetkých, ktorí si chcú vytvoriť vlastný názor na základe faktov. K tomuto účelu môžu slúžiť dáta, ktoré sú verejne dostupné a ktoré sa budú verejnosti poskytované v jednoduchej a zrozumiteľnej podobe.

Jeden z príkladov takýchto projektov zaoberajúci sa zobrazením verejne dostupných dát je práve projekt Zastupko.cz[1]. Tento projekt sa zaoberá zobrazením a analýzou dát z hlasovania zastupiteľstiev v Českej republike. Najskôr bol projekt mienený čisto pre mesto Brno, no neskôr sa rozšíril a pridal spracovanie a zobrazovanie dát aj pre ďalšie kraje a mestá. Jedná sa o verejne dostupný webový informačný systém, kde si ktokoľvek vie pozrieť, ako hlasovali jednotliví zastupitelia municipalít, k akej politickej strane prislúchajú alebo aká je ich dochádzka na zasadaniach. Poskytuje tiež niektoré detailnejšie analýzy, ako napríklad aj porovnanie dvoch konkrétnych zastupiteľov či koherenciu hlasovaní. Na základe týchto objektívnych dát si vie užívateľ na jednom mieste spraviť obraz o niektorých politických stranách či jej členoch.

Ako každý systém, je dôležité, aby obsahoval robustné API. Dobré zobrazenie dát je veľmi dôležité, avšak je tiež kľúčové zo serverovej strany, aby rozhranie ponúkalo širokú škálu dobre spracovaných dát, ktoré je možné zobraziť. Táto práca sa preto zameriava na serverové rozhranie tohto informačného systému pre spracovanie a zobrazovanie dát z hlasovania zastupiteľstiev v Českej republike. Cieľom tejto práce je vytvorenie novej architektúry serverovej časti a jej rozhrania spolu s odladením chýb z analyzovanej verzie tohto systému a implementovanie nových požiadaviek na systém. Ďalším dôležitým cieľom je integrácia špecifikácie aplikačného rozhrania serverovej strany pre lepšiu testovateľnosť a dokumentáciu API.

Najskôr sú v teoretickej časti rozobrané náležitosti potrebné pre pochopenie kontextu tejto práce. Najskôr sú vysvetlené náležitosti týkajúce sa fungovania samosprávnych orgánov v kapitole 2, kde je rozobraná problematika, základné pojmy a fakty týkajúce sa zastupiteľstiev krajov a obcí v Českej republike. Táto kapitola je dôležitá pre pochopenie fungovania samotného hlasovania a prijímania uznesení. V tej súvislosti sú ďalej vysvetlené kľúčové záležitosti k štruktúre a získavaniu dát z municipalít, ktoré sú informačným systémom

[1] Dostupné z: zastupko.cz

spracované a zobrazované. Ďalšou kľúčovou znalosťou je porozumenie technológiám pre vývoj webových aplikácií. Zameranie je kladené najmä na porovnanie existujúcich architektúr webovej aplikácie, technológie serverového rozhrania RESTful API, objektovo-relačné mapovanie (ORM) a vytváranie dokumentácie aplikačného rozhrania. Tieto technológie sú popísané v kapitole 3 a znalosti z nich použité pri samotnom návrhu a implementácii. Preto je aj potrebné ich preskúmanie a vysvetlenie ich využitia v súvislosti so skúmaným riešením. Nasledujúca kapitola 4 obsahuje analýzu informačného systému Zastupko.cz. Zámer je kladený najmä na analýzu doterajšej implementácie serverovej časti a spracovania požiadaviek spolu s komunikáciou s databázou. Analyzované sú tiež súčasné problémy tohto riešenia. Táto kapitola je dôležitá pre bližšie priblíženie a rozobratie implementácie analyzovaného systému, na ktorom táto práca stavia.

Tématický po analýze systému hlasovania zastupiteľstiev nasleduje kapitola 5 s návrhom novej architektúry, návrhom riešení problémov analyzovaných v predošlej kapitole a návrh integrovania špecifikácie API. V tejto kapitole sú opísané hlavné zmeny, ktoré budú vykonané na informačnom systéme. Kapitola 6 obsahuje ďalej implementáciu týchto navrhovaných zmien a popis problémov, s ktorými sa bolo potrebné počas implementácie vysporiadať a ich následné riešenie. Ďalej sú opísané technológie použité počas implementácie architektúry a integrácie špecifikácie aplikačného rozhrania. Implementované riešenie je následne otestované pomocou upravených testov, prítomných v existujúcom riešení. Testy sú rozšírené podľa upravených funkcionalít a odpovedí z koncových bodov. Podrobnejší popis testovania je v kapitole 7. Ako posledná je kapitola so záverom 8, kde sú zhrnuté dosiahnuté výsledky vytvoreného riešenia. V kapitole sú tiež navrhnuté prípadné námety na budúce zmeny a optimalizácie serverovej časti tohto informačného systému.

Kapitola 2

Zastupiteľstvá a ich funkcia

Pre účely tejto práce je dôležité pochopiť, ako fungujú zastupiteľstvá pre samosprávy a prečo je dôležité zaoberať sa dátami z nich. Je taktiež žiaduce vysvetliť samotnú hierarchiu, ktorá vzniká v oblasti štátnej správy krajov a obcí v Českej republike. V tejto kapitole budú vymedzené základné pojmy, ktoré súvisia s touto prácou.

Dôležitým pojmom je **verejná správa**, ktorá predovšetkým značí správu verejných záležitostí vykonávanú pre tú činnosť určenými orgánmi. Existuje viacero pohľadov, ako môže byť tento pojem chápaný. Jeden z pohľadov naznačuje isté rozdelenie na subsystémy a to na štátnu správu a samosprávu [30, 33].

Ako už z názvu vyplýva, **samosprávou** sa v najvšeobecnejšom zmysle rozumie predovšetkým samostatné spravovanie záležitostí, ktoré sa bezprostredne dotýkajú jej obyvateľov [30]. Jedná sa o prejav decentralizácie, kedy je určitá oblasť verejnej správy prenesená na sám seba spravujúci orgán, ktorý ju následne vykonáva sám [10]. Ako je poznamenané v práci [24]: „Samospráva je verejná správa vykonávaná inou inštitúciou ako štátom“.

Ďalším pojmom je **štátna správa**, ktorá je kľúčovou zložkou verejnej správy a neoddeliteľnou súčasťou každého štátom organizovaného spoločenstva. Predstavuje tak akési jadro celého systému verejnej správy [23]. Z hľadiska svojej povahy je štátna správa špecifickým typom spoločenského usporiadania, ktoré je vykonávané samotným štátom, respektíve jeho orgánmi. Ciele, ktoré majú byť dosiahnuté a ktoré sú zverené štátnej správe, sú stanovené právnym poriadkom [46].

2.1 Územné celky

Zákon *č. 1/1993* [11] člení územie Českej republiky na kraje, ktoré sú vyššími územnými samosprávnymi celkami a na obce, ktoré sú základnými územnými samosprávnymi celkami. Každá obec a vojenský újazd sú súčasťou niektorého z krajov [42].

Krajské zastupiteľstvá

Vzhľadom k zvolenému systému českej verejnej správy vykonávajú kraje svoje kompetencie v samostatnej a prenesenej pôsobnosti [9]. Konkrétne vymedzenie všetkých 14 krajov vrátane hlavného mesta Praha a jeho území je obsahom ústavného zákona *č. 347/1997 Sb.* [12]. Informácie v nasledujúcej časti sa vzťahujú na spomínaný zákon. V rámci samostatného pôsobenia majú kraje také kompetencie, ktoré sú stanovené v zákone o krajoch [14], medzi ktoré sa radia napríklad vydávanie všeobecne záväzných vyhlášok, územný rozvoj, zákonodarná iniciatíva a iné.

Občanom kraja je podľa §12 fyzická osoba s občianstvom v Českej republike, ktorá má trvalý pobyt v obci alebo na území vojenského újazdu na území daného kraja [33].

Základnými orgánmi krajskej samosprávy sú predovšetkým zastupiteľstvo, ďalej rada, hajtman, krajský úrad a zvláštne úrady kraja [33]. Ďalej sú opísané jednotlivé orgány krajskej samosprávy.

Zastupiteľstvám kraja a ich orgánom sa venuje predovšetkým zákon [14] v hlave 4. Počet členov zastupiteľstva sa určuje podľa počtu obyvateľov:

- do 600 000 obyvateľov 45 členov,
- nad 600 000 do 900 000 obyvateľov 55 členov,
- nad 900 000 obyvateľov 65 členov.

Právomoci zastupiteľstva sú potom ďalej popísané v §35. Zastupiteľstvo rozhoduje vo veciach patriacich do samostatnej pôsobnosti. Vo veciach prenesenej pôsobnosti zastupiteľstvo rozhoduje len v prípade, že tak stanoví zákon. Medzi niektoré vyhradené právomoci patrí podľa §35 napríklad predkladať návrhy zákonov Poslaneckej snemovni, vydávať všeobecne záväzné vyhlášky kraja, voliť zástupcov kraja do regionálnych rád regiónov súdržnosti či voliť a odvolávať hajtmana, zástupcu hajtmana a ďalších členov rady.

Podľa §43 je o priebehu zasadania zastupiteľstva, ktoré sa zasadá minimálne každé tri mesiace, vytvára zápis. Tento zápis musí obsahovať údaj o počte prítomných členov zastupiteľstva, schválený program jednania, priebeh a výsledok hlasovania a prijaté uznesenia. K tomu, aby bolo uznesenie, rozhodnutie alebo voľba platná, je potrebný súhlas nadpolovičnej väčšiny všetkých členov zastupiteľstva. Návrh programu jednania zastupiteľstva pripravuje a predkladá zastupiteľstvu k schváleniu rada. O zaradení návrhov v ďalších bodoch programu, prednášaných v priebehu zasadania, rozhoduje samotné zastupiteľstvo.

Ďalším už spomínaným orgánom je **rada**, ktorá je popísaná v Hlave IV diel 2 §57 - 60. Rada je výkonným orgánom kraja v oblasti samosprávnej pôsobnosti. Pri výkone svojej pôsobnosti zodpovedá rada. Rada môže rozhodovať vo veciach prenesenej pôsobnosti, ak tak stanoví zákon. Radu tvorí hajtman, zástupca hajtmana a ďalší členovia rady. Počet členov rady sa delí podobne podľa počtu obyvateľov kraja, kde pri počte obyvateľov do 600 000 má rada 9 členov a 11 členov nad tento počet obyvateľov.

Schádzanie členov rady nie je nejak pravidelné, ale je vykonávané podľa potreby. Schôdze rady sú neverejné, ale môže prizvať k jednotlivým bodom svojho jednania aj ďalšieho člena zastupiteľstva alebo iné osoby. Taktiež, ako pri zastupiteľstve, je o schôdze rady vytváraný zápis s rovnakými náležitosťami.

Ako už bolo spomínané, rada pripravuje návrhy a podklady pre jednanie zastupiteľstva a zabezpečuje plnenie prijatých uznesení. K ďalším úlohám rady podľa §59 patrí napríklad zabezpečovať hospodárenie podľa schváleného rozpočtu, či vydávať nariadenie kraja.

Krajský úrad, podľa dielu 5 §66, plní úlohy v samostatnej pôsobnosti uložené zastupiteľstvom a napomáha činnosti výborov komisie. Zastupiteľstvo ani rada nemôžu na krajský úrad prenášať úlohy, ktoré sú im vyhradené zákonom. Ďalšími úlohami krajského úradu je podľa §67 napríklad zabezpečovať výstavbu a prevádzkovanie informačného systému či kontrolu dodržovania opatrení vlády v činnosti okresných úradov.

Ďalšie zvláštne orgány zriaďuje hajtman pre výkon prenesenej pôsobnosti, pokiaľ tak stanoví zákon.

Obecné zastupiteľstvá

Primárne legislatívne ustanovenia týkajúce sa obcí a ich orgánov v Českej republike sú popísané v prvej časti zákona *č. 128/2000 Sb.* [14]. Existujú rôzne druhy obcí, respektíve môžu niesť aj iný názov než obec. Prvou možnosťou je „městys“, ktorou sa môže obec stať na vlastnú žiadosť, pokiaľ je to schválené vládou. Ďalším druhom je mesto, u ktorého je podmienka, že obec musí mať minimálne 3 000 obyvateľov [33]. Obec môže taktiež nadobúdať názov štatutárne mesto. Obce s týmto označením sú napríklad Most, Olomouc či Brno. Orgány sú v tomto prípade podobné ako u krajov. Základným orgánom obce je tiež zastupiteľstvo. Ďalšími orgánmi sú rada, starosta, obecný úrad a zvláštne orgány obce. Orgány krajov sú popísané v Hlave 4 spomínaného zákona v §67 až 116.

Do samostatnej pôsobnosti obce patria najmä záležitosti uvedené v spomínanom zákone v §84, 85 a 120. Do samostatnej pôsobnosti obce patrí spravovanie záležitostí, ktoré sú v záujme obce a jej občanov, pokiaľ nie sú zverené zákonom krajom alebo pokiaľ nejde o výkon prenesenej pôsobnosti.

Dôležitým orgánom obce je aj v tomto prípade **Zastupiteľstvo**, ktoré je podobne definované ako v prípade kraja. Je však odlišný počet členov zastupiteľstva v závislosti od počtu obyvateľov nasledovne:

- do 500 obyvateľov 5 - 9 členov,
- nad 500 do 3 000 obyvateľov 7 - 15 členov,
- nad 3 000 do 10 000 obyvateľov 11 - 25 členov,
- nad 10 000 do 50 000 obyvateľov 15 - 35 členov,
- nad 50 000 do 150 000 obyvateľov 25 - 45 členov,
- nad 150 000 obyvateľov 35 - 55 členov.

Ak nie je v obci zastupiteľstvo, stanoví počet členov zastupiteľstva obce pre účely volieb krajský úrad. Ak nie je v mestskom obvode územne členeného štatutárneho mesta zastupiteľstvo mestského obvodu, v mestskej časti takého mesta zastupiteľstvo mestskej časti, stanoví počet členov tohto zastupiteľstva pre účely volieb magistrátu. Zastupiteľstvo obce rozhoduje vo veciach patriacich do samostatnej pôsobnosti obce spomínaných v úvode tejto podkapitoly a v uvedenom zákone (hlava 2 diel 1 §35). Zastupiteľstvo obce môže zriadiť ako svoje iniciatívne a kontrolné orgány komisie a výbory, ktoré dohliadajú na jej činnosť.

Podobne je ďalším orgánom **rada obce**. Tvorí ju starosta, zástupca/zástupcovia starostu a ďalší členovia rady. Počet členov rady je nepárny počet a tvorí najmenej 5 a najviac 11 členov, pričom nesmie prekročiť tretinu počtu členov zastupiteľstva obce. Funkciu rady obce plní starosta v obci, v ktorej nie je zriadená rada obce. Podľa §103 starosta zastupuje obec navonok. Úkony, ktoré vyžadujú schválenie zastupiteľstva obce, prípadne rady obce, môže starosta vykonať iba po ich predchádzajúcom schválení. Právomoci a úlohy rady obce sú podobné ako v prípade kraja.

2.2 Dáta o činnosti zastupiteľstiev

Okrem samotného popisu fungovania zastupiteľstiev je dôležitejšie zaoberať sa dátami, ktoré poskytujú a do akej miery. Ako bolo spomínané v kapitolách 2.1 a 2.1 o krajoch a obciach,

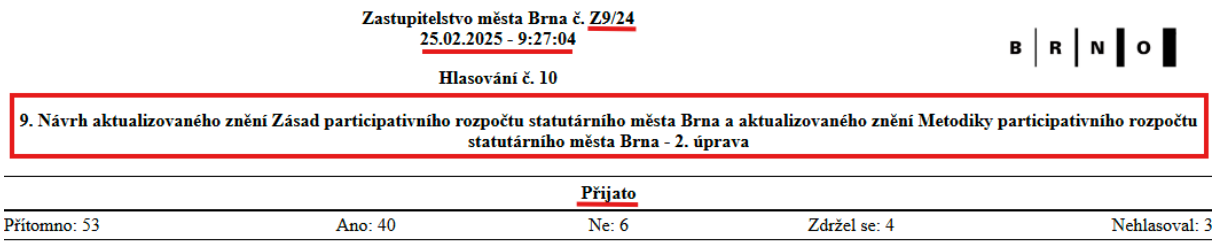
z jednotlivých zasadnutí zastupiteľstiev a orgánov je vytváraný zápis o priebehu zasadania, ktorý obsahuje najmä počet prítomných členov, schválený program jednania, priebeh a výsledok hlasovania. Každé zasadnutie má priradené body jednania, o ktorých sa diskutuje a hlasuje [26]. K jednému bodu zasadania môže prebiehať aj viacero hlasovaní [28]. Zastupiteľstvá okrem tohto zápisu obsahujú aj iné dáta spojené s ich orgánmi, avšak pre účel tejto práce sú najpodstatnejšie dáta súvisiace s hlasovaním, zasadaniami, samotnými členmi a ich prítomnosťou na zasadaniach.

Najskôr je však dôležité pochopiť, ako sú spomínané dáta sprístupňované. Definíciu pojmu o otvorených dátach priniesol zákon č. 298/2016 Sb. [13]. Ide o informácie, ktoré sú zverejňované takým spôsobom, že umožňujú diaľkový prístup v otvorenom a strojovo čitateľnom formáte [29]. Podľa zdroja [41] sa definujú otvorené dáta ako na súkromie neobmedzujúce a nedôverné dáta, ktoré sú vytvorené z verejných peňazí a bez obmedzenia ich použitia a distribúcie. Pre krajské úrady a všeobecné úrady obcí s rozšírenou pôsobnosťou existuje podľa zákona č. 261/2021 [16] povinnosť metadáta informácií zverejňovať na úradných tabuliach ako otvorené dáta [45].

Otvorené dáta sprístupňuje *Národní katalóg otevřených dat* (NKOD) [32] podľa ktorého sú dáta otvorené, pokiaľ sú voľne prístupné na webe ako dátový súbor, ktorý je možné stiahnuť v strojovo čitateľnom a otvorenom formáte (napr. XML, CSV, JSON) [29]. Nie je však definovaný žiadny predpísaný formát týchto dát, čo zhoršuje aj ich spracovanie. Priebeh hlasovania môže byť zapísaný v PDF formáte či v oskenovanej podobe, čo spôsobuje problém tieto dáta strojovo spracovať. Väčšinou sa tento spôsob využíva u zastupiteľstiev v menších obciach. Taktiež nie všetky dáta zo zastupiteľstiev obcí Českej republiky je možné sprístupniť v NKOD [29, 33]. Časť zastupiteľstiev obcí však svoje dáta sprístupňuje na svojich verejných stránkach, kde sa vyskytujú aj v strojovo čitateľnom formáte buď ako HTML alebo je občas prístupná dátová sada v podobe JSON či CSV.

Samotné hlasovacie dáta predstavujú záznam o tom, ako bolo hlasované pri prerokovaní rôznych bodov programu zasadnutia. Príklad záznamu jedného hlasovania je možné vidieť na obrázku 2.1. Štandardne tieto dáta obsahujú nasledujúce informácie:

- **Identifikácia hlasovania** – dátum a číslo hlasovania v rámci zasadnutia.
- **Predmet hlasovania** – stručný popis prerokovanej záležitosti (napr. schválenie rozpočtu alebo zmeny územného plánu).
- **Výsledok hlasovania** – počet hlasov pre jednotlivé možnosti hlasovania pre, proti a zdržal sa.



Zastupitelstvo města Brna č. Z9/24
25.02.2025 - 9:27:04

B | R | N | O

Hlasování č. 10

9. Návrh aktualizovaného znění Zásad participativního rozpočtu statutárního města Brna a aktualizovaného znění Metodiky participativního rozpočtu statutárního města Brna - 2. úprava

Přijato

Přítomno: 53	Ano: 40	Ne: 6	Zdržel se: 4	Nehlasoval: 3

Obr. 2.1: Ukážka hlavičky hlasovacieho protokolu Zastupitelstva mesta Brno č. Z9/24, bod č.9, hlasovanie 10[2]. Zvýraznené časti predstavujú dôležité časti protokolu ako číslo zasadania, datum, čas konania, predmet a výsledok hlasovania

Napriek tomu, že zaznamenávať výsledok je povinnou súčasťou zápisu, nekladie sa požiadavka na zaznamenávanie hlasov jednotlivých členov zastupiteľstva [70]. Samozrejme, niektoré väčšie zastupiteľstvá, ako napríklad v prípade mesta Praha[3] sprístupňujú aj hlasy jednotlivých členov. Čo sa týka hlasovania, je taktiež dôležité poznamenať, že napriek tomu, že rokovania zastupiteľstva sú verejné, samotné hlasovania môžu byť tajné [28].

Prehľadnou formou, ako môžu byť dáta o hlasovaní dostupné, je vo forme hlasovacieho protokolu, napríklad v prípade zastupiteľstva mesta Brno[4]. Táto forma nie je predpísaná zákonom, ale niektoré mestá s týmto termínom operujú. Hlasovací protokol je možné definovať ako dokument obsahujúci údaje týkajúce sa priebehu zasadania a jednotlivých hlasovaní. Typicky sa jedná o výstup systému pre zaznamenávanie hlasovania, vďaka čomu môžu obsahovať podrobnejšie dáta ako samotný zápis [33]. Majoritnými príkladmi týchto systémov sú H.E.R. Systém[5], MINISTR[6] a Usnesení.cz[7] [29, 33]. Zastupitelia zadávajú svoje hlasy prostredníctvom hlasovacieho zariadenia (obrázok 2.1) ako napríklad v prípade H.E.R. Systému. Zákon taktiež nevymedzuje spôsob vytvárania záznamu z hlasovania, takže zastupiteľstvá môžu pristupovať k zaznamenávaniu hlasov rôznym spôsobom, vrátane ručného sčítavania hlasov [28]. Použitie hlasovacích systémov je skôr typické pre väčšie mestá, najmä aj kvôli väčším finančným prostriedkom [67].

Obr. 2.2: Hlasovacie zariadenie H.E.R. Systému[9]obsahujúce tlačidla pre zvolenie hlasovacej možnosti (pre, proti, zdržal sa) a tlačidlá pre zapnutie mikrofónu, žiadosť o slovo či technickú poznámku.

Okrem hlasovania a výsledkov hlasovania je zákonom stanovené, že zápis musí taktiež obsahovať zoznam prítomných členov [70]. Údaje o účasti poslancov na zasadnutiach zastupiteľstva predstavujú ďalší dôležitý aspekt hodnotenia činnosti volených zástupcov. Podľa odporúčania [70] vydaného Ministerstvom vnútra Českej republiky by sa mali uvádzať aj počet ospravedlnených a neospravedlnených členov neprítomných na zasadaní. Zo súčtov týchto údajov je možné zistiť celkový počet členov zastupiteľstva obce, ktorý je dôležitý u prípadného skúmania platnosti uznesenia.

[2]Prevzaté z: apl.brno.cz/protokoly-zmb/zmb-z9-24/20250225-10-34024.html
[3]Príklad dát zo zasadania: praha.eu/vysledky-hlasovani/detail/103774
[4]Protokol zo zasadania mesta Brno: apl.brno.cz/protokoly-zmb/zmb-z9-24/
[5]bitest.cz/hlasovaci-system/
[6]ministr.cz
[7]usneseni.cz
[9]Prevzaté z: https://bitest.wphost02.ewdev.cz

Aj pri údajoch o dochádzke platí, že spôsob ich zverejňovania sa líši medzi jednotlivými obcami. Väčšie mestá majú tendenciu sprístupňovať tieto informácie prehľadnejšie a systematickejšie než menšie obce, kde dochádzkové údaje môžu byť dostupné len v rámci úplných zápisníc bez ďalšej štruktúry, aj napriek spomínanému odporúčaniu.

2.3 Porovnanie so slovenskou samosprávou

Táto podkapitola porovnáva fungovanie a dáta zastupiteľstiev v porovnaní so Slovenskou republikou. Porovnanie týchto dvoch štátov je vhodné porovnať z hľadiska toho, že tieto dva štáty relatívne nedávno tvorili jeden celok do roku 1993 [54]. Je teda do istej miery pravdepodobné, že budú mať tento systém samospráv a zastupiteľstiev veľmi podobný a vieme nájsť isté znaky v dátach, ktoré samosprávy zverejňujú. Porovnanie v tejto podkapitole je zhrnuté v tabuľke 2.1 a ďalej rozpísané v zvyšku podkapitoly.

Čo sa týka štruktúry samospráv, v oboch štátoch je využité rovnaké členenie na kraje, ktoré sa delia na okresy a ďalej na samotné obce. Rozdiel v tomto prípade nájdeme iba v samotnom počte samospráv. Napriek podobnosti členenia samospráv štátov vieme nájsť isté zásadné rozdiely v spôsobe hlasovania a prijímania uznesení [65, 66]. Konkrétnejšie rozdiely sú opísané nižšie v tejto podkapitole.

Komparáciou združení krajov v Česku a Slovensku sa venuje práca [20]. Zákon [15] v prípade Českej republiky a zákon [56] v rámci Slovenskej udáva, že kraj má právo rozvíjať spoluprácu s územnými samosprávnymi celkami iných štátov, z čoho vyplýva, že už v období prijímania týchto zákonov sa dala predpokladať spolupráca medzi krajmi a to nie len v rámci jedného štátu. Hlavnou úlohou v oboch prípadoch je obhajovať a presadzovať spoločné záujmy a práva krajov vo vzťahu k ústredným orgánom štátnej správy, ale aj v rámci fungovania Európskej únie. Právna úprava v Českej republike aj na Slovensku zakotvuje obecné zastupiteľstvo ako hlavný orgán samosprávy obce. Líšia sa samozrejme kritériá počtu zastupiteľov, kde v oboch prípadoch je počet určený počtom obyvateľov.

Pre účely tejto práce je však podstatný spôsob hlasovania a prijímania uznesení zastupiteľstiev. Týmto rozdielom sa venuje aj práca [49], podľa ktorej je jeden z rozdielov frekventovanosť zasadnutí. Uvádza, že zasadnutia zastupiteľstva sa majú konať podľa potreby, no v Českej republike podľa zákona o obciach sa má schádzať najmenej raz za tri mesiace. Na Slovensku zákonná podmienka určuje zasadnutie minimálne jedenkrát za dva mesiace. Jedno z porovnaní taktiež hovorí o kritériách platnosti hlasovania, kde v Česku pre platné uznesenia je potrebný súhlas nadpolovičnej väčšiny všetkých členov zastupiteľstva. Naopak, na Slovensku sú podmienky na prijímanie uznesení nastavené miernejšie. Spôsobilosť prijímať uznesenia je naviazaná na súhlas nadpolovičnej väčšiny prítomných členov zastupiteľstva [49].

Ďalším výrazným rozdielom je, že české zastupiteľstvo môže prijímať len všeobecne záväzné vyhlášky obce v samostatnej pôsobnosti. Právna úprava na Slovensku to nerozlišuje a zastupiteľstvo obce prijíma všeobecne záväzné nariadenia obce v samostatnej aj prenesenej pôsobnosti. V Českej republike prijíma nariadenia v prenesenej pôsobnosti obce samotná rada [20, 49].

Čo sa týka samotných otvorených dát z hlasovania zastupiteľstiev, vieme v Českej republike nájsť dátové sady vďaka už spomínanému NKOD[10] v predošlej kapitole. Obdobou v rámci Slovenska je Národný katalóg otvorených dát[11], ktorý rovnako obsahuje sady otvore-

[10] data.gov.cz
[11] data.slovensko.sk

ných dát zo samospráv a zastupiteľstiev. Česká republika má v tomto smere mierne náskok, pretože niektoré mestá, ako Praha[12] či Brno[13], ponúkajú viac otvorených dát vo formáte, ktorý je vhodný na ďalšiu analýzu (napr. strojovo čitateľné formáty ako je JSON). Slovensko síce tiež napreduje v oblasti otvorených dát, avšak dostupnosť údajov o hlasovaní zastupiteľstiev je často fragmentovaná a závislá od konkrétnej samosprávy. Na slovenskej strane v Národnom katalógu otvorených dát taktiež momentálne nenájdeme dátovú sadu, ktorá by obsahovala dáta zo zasadnutia zastupiteľstva a priebehu jeho hlasovania. Tieto dáta vieme nájsť až v lokálnych katalógoch dát konkrétnych miest, ako napríklad pre hlavné mesto Bratislava, kde sú obsiahnuté hlasy jednotlivých členov zastupiteľstva ku konkrétnym bodom [40].

Je dôležité poznamenať, že môžeme dáta a dátové sady zo zastupiteľstiev porovnávať aj vďaka tomu, že obidva štáty majú podobnú štruktúru zastupiteľstva a priebeh hlasovania, z ktorého vzniká zápis, ako bolo opísané v predošlej podkapitole. V obidvoch prípadoch sa zastupitelia obce stretávajú a rokujú počas zasadnutí, kde sú predložené body zasadnutia, ku ktorým sa hlasuje. Znak spolupráce je možné nájsť aj v spôsobe získavania hlasovacích dát. Zastupiteľstvá pre zaznamenávanie hlasovania využívajú špecifické systémy, z ktorých sú vytvorené protokoly o výsledku hlasovania. Jedným z týchto systémov je už spomínaný H.E.R. Systém vyrábaný v Slovenskej republike [33] a používaný aj v zastupiteľstve mesta Bratislava [14].

V Slovenskej republike sa o istú centralizáciu záznamov z hlasovania zastupiteľov pokúša projekt zastupitelstvo.sk, kde z dostupných informácií je na webe archivovaných a vysielaných záznamov zo zasadaní všetkých krajských miest a dokopy zahŕňa 34 miest zo všetkých krajov [18]. Keďže ide iba o videozáznamy, nie sú prístupné analytické zhrnutia výsledkov konkrétnych zasadaní ani ich samotné zápisy, no ponúkajú možnosť vyhodnotiť dodatočné informácie vyplývajúce z videozáznamu, ktoré čistý zápis zo zasadania neobsahuje, ako napríklad konkrétnu verbálnu rozpravu medzi zastupiteľmi. Ako podobný projekt snažiaci sa o istú centralizáciu dát zastupiteľstiev môžeme v Českej republike označiť aplikáciu UZOb, ktorá zahŕňa dáta z približne 306 obcí [57]. Obsahuje prevažne zápisy zo zasadaní obcí a oproti zastupitelstvo.sk neobsahuje žiadne videozáznamy. Na druhej strane, podľa dostupných dát, UZOb má väčší rozsah, čo sa týka počtu miest a obcí, z ktorých sprístupňuje dáta. Na obidvoch stranách je však spoločným znakom snaha o sprístupnenie aktuálnych dát o hlasovaní zastupiteľstiev ich občanom.

Napriek rozdeleniu týchto krajín a rozdielnemu vývoju stále pretrváva vzájomná spolupráca nielen na medzištátnej úrovni, ale aj v rámci miest a obcí. Z hľadiska regionálnej spolupráce predstavuje Vyšehradská skupina (V4) priestor na výmenu osvedčených praktík v oblasti digitálnej transparentnosti [43]. Spoločná historická a kultúrna blízkosť krajín V4, vrátane Česka a Slovenska, vytvára predpoklady pre vytvorenie spoločných štandardov alebo metodických rámcov pri publikovaní dát o činnosti samospráv.

[12] praha.eu/zastupitelstvo
[13] www.brno.cz/zastupitelstvo-mesta-brna
[14] www.aspartner.sk/referencie

Oblasť	Česká republika	Slovenská republika
Frekvencia zasadnutí zastupiteľstiev	Minimálne raz za tri mesiace	Minimálne raz za dva mesiace
Podmienky platnosti uznesení	Nadpolovičná väčšina všetkých členov zastupiteľstva	Nadpolovičná väčšina prítomných členov
Prijímanie nariadení	Obecné zastupiteľstvo prijíma nariadenia v samostatnej pôsobnosti; rada v prenesenej pôsobnosti	Zastupiteľstvo prijíma všeobecne záväzné nariadenia v samostatnej aj prenesenej pôsobnosti
Dostupnosť hlasovacích dát	Rozsiahlejšie otvorené dáta dostupné aj v NKOD, strojovo spracovateľné formáty (JSON) – napr. Praha, Brno	Údaje v NKOD menej dostupné; väčšinou lokálne katalógy, napr. Bratislava
Iniciatívy na centralizáciu dát	UZOb – databáza zápisov z 306 obcí (textové zápisy)	zastupitelstvo.sk – videozáznamy z rokovaní 34 miest (bez analytických údajov)

Tabuľka 2.1: Zhrnutie podstatných rozdielov fungovania zastupiteľstiev a dostupnosti dát v Českej a Slovenskej republike. Hlavné rozdiely sú okrem počtu krajov a obcí hlavne vo frekvenciách zasadania a podmienkach platnosti uznesení.

Kapitola 3

Serverová časť webových aplikácii

Samotná webová aplikácia je, ako uvádza zdroj [38], zložená z troch hlavných častí, a to klientská časť, serverová časť a databáza. Hlavnou úlohou klientskej časti je spracovanie užívateľských vstupov a správa užívateľského rozhrania. Serverová časť spracováva dáta a poskytuje pokročilé funkcie prvkom na strane klienta. Z databázovej časti sa následne získavajú a ukladajú dáta.

Serverová časť webových aplikácií tvorí teda neoddeliteľnú súčasť každej modernej webovej aplikácie. V tejto kapitole sa budeme zaoberať základnými princípmi a modelmi serverovej architektúry, technológiami a prístupmi pri vývoji serverových riešení. Pozornosť bude venovaná aj aspektom výkonnosti a dokumentácie, ktoré majú zásadný vplyv na kvalitu výslednej aplikácie.

3.1 Architektúry serverových aplikácii

Pri vývoji moderného softvéru môže výber správnej architektúry vytvoriť dobrý základ pre škálovateľnosť, flexibilitu a celkový výkon aplikácie. Rôzne architektúry formujú nielen to, ako je kód organizovaný, ale aj to, ako služby interagujú, škálujú a udržiavajú sa v dlhšom časovom rozmedzí. V nasledujúcej podkapitole budú rozobrané a porovnané niektoré architektúry používané pri vývoji serverovej časti aplikácie. Každá architektúra prináša svoje výhody, limitácie a špecifické prípady použitia.

Ako uvádza práca [44] existujú isté architektonické charakteristiky, ktoré určujú jeho štruktúru, správanie a kvalitatívne vlastnosti. Ich cieľom nie je splniť všetky požiadavky naraz, ale nájsť rovnováhu medzi očakávaniami, obmedzeniami a praktickými limitmi konkrétneho systému. Tieto charakteristiky spolu vytvárajú základ pre výber vhodnej architektúry a dlhodobú udržateľnosť aplikácie. Medzi kľúčové architektonické charakteristiky patrí:

- **škálovateľnosť** – schopnosť systému zvládať rastúcu záťaž bez straty výkonu,
- **spoľahlivosť** – konzistentné správanie systému aj pri výpadkoch či poruchách,
- **efektivita výkonu** – rýchlosť, účinnosť a optimálne využitie zdrojov,
- **bezpečnosť** – ochrana dát, overovanie používateľov a odolnosť voči hrozbám,
- **udržiavateľnosť** – jednoduché testovanie, rozšírenie a prispôsobenie systému,
- **rozšíriteľnosť** – možnosť doplnenia nových funkcií bez narušenia existujúcich,

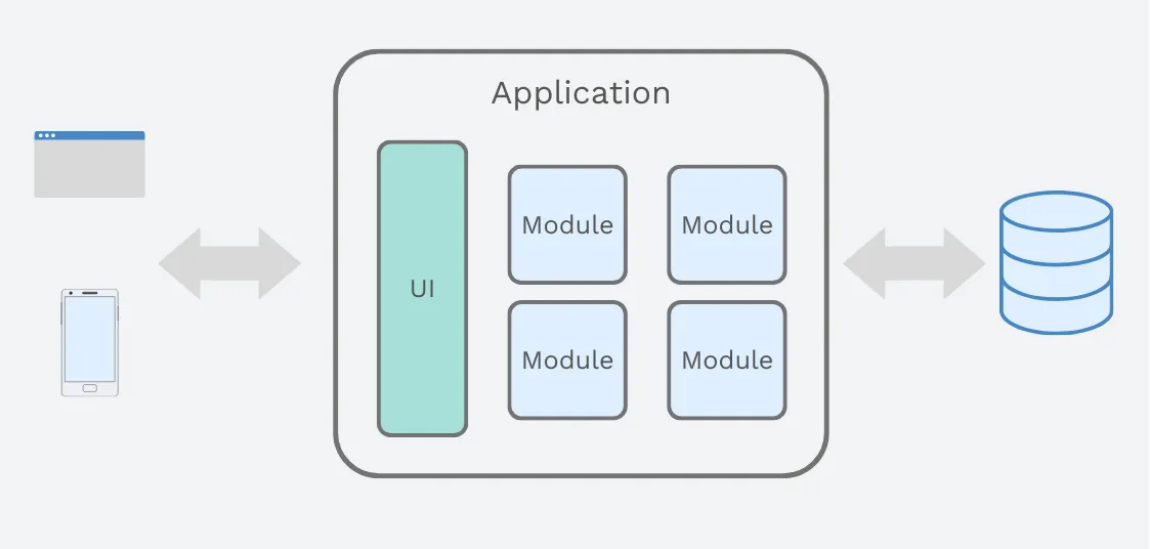
- **testovateľnosť** – schopnosť efektívne testovať jednotlivé časti systému,
- **pozorovateľnosť** – možnosť analyzovať vnútorné správanie systému pomocou výstupov a logovacích súborov.

Monolitická architektúra

Monolitická architektúra predstavuje tradičný, jednovrstvový prístup k vývoju aplikácií, v ktorom sú všetky komponenty, ako užívateľské rozhranie (UI), biznis logika aj prístup k dátam, zabalené a nasadené ako jeden celok [58]. Schéma monolitickej architektúry je zobrazená na obrázku 3.1

Najväčšími výhodami podľa zdroja [7] je jednoduchosť tejto architektúry. V porovnaní s distribuovanými aplikáciami je jednoduchšia na testovanie, nasadzovanie, ladenie aj monitorovanie. Všetky dáta sú uložené v jednej databáze, nie je potrebná ich synchronizácia, a všetka komunikácia prebieha v rámci jedného procesu, čo je rýchle a spoľahlivé. Monolitická architektúra je prirodzeným a častým počiatočným prístupom pri vývoji aplikácie. Všetka logika beží v jednom procese a aplikácia sa dá jednoducho štruktúrovať pomocou tried, funkcií a menných priestorov v preferovanom jazyku vývojárov.

Ako uvádzajú zdroje [7, 27] monolitický prístup je často výhodný v počiatočných fázach projektu, keď je aplikácia malá a tím vývojárov ešte nie je rozsiahly. S rastom aplikácie môže táto architektúra spôsobovať problémy, ako napríklad ťažkosti s údržbou, pomalé nasadzovanie zmien alebo obmedzené škálovanie (pretože sa všetko musí spúšťať a nasadzovať naraz). V modernom vývoji sa preto často uprednostňujú iné architektúry, ktoré umožňujú oddelený vývoj, škálovanie a nasadzovanie jednotlivých častí systému.



Obr. 3.1: Schéma monolitickej architektúry[2], implementácia užívateľského rozhrania a aplikačnej logiky sú implementované na jednom mieste, ktoré komunikuje s klientom (vľavo) a databázou (vpravo).

[2]Prevzaté z: miro.medium.com/v2/resize:fit:4800/format:webp/1*yf4RsFWOQvidhSBqCLGrIw.png

Architektúra orientovaná na služby (SOA)

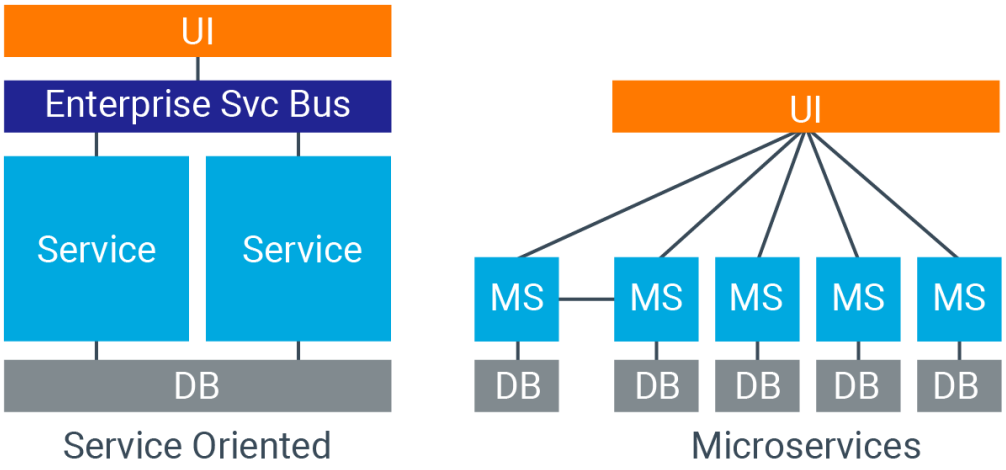
Jedná sa o architektúru, ktorá organizuje funkcie aplikácie do samostatných služieb. Tieto služby komunikujú medzi sebou pomocou jasne definovaných rozhraní (zvyčajne API). Každá služba zastrešuje konkrétnu obchodnú funkcionalitu (napr. autentifikáciu používateľa, spracovanie platieb či správu dát) a je možné ju používať nezávisle [31, 62]. Podľa zdroja [39] sú v modeli SOA definované tri role, a to poskytovateľ služby, sprostredkovateľ a spotrebiteľ.

Rovnaký zdroj uvádza istú charakteristiku, ktorá môže byť braná ako výhoda, a to znovupoužiteľnosť, kde služby možno opakovane využiť v rôznych aplikáciách, čo šetrí čas a zaisťuje jednotnosť. Služby sa dajú vyvíjať a aktualizovať nezávisle, čo zvyšuje flexibilitu. Časti aplikácie sa môžu do určitej miery škálovať samostatne, čo je výhodné pri náročných podnikových aplikáciách.

Ďalší zdroj [64] uvádza aj isté nevýhody tejto architektúry, ako napríklad komunikácia medzi službami cez sieť môže spôsobovať oneskorenie (aj keď menšie ako pri architektúre mikroslužieb). Komplexnosť architektúry si vyžaduje silné riadiace mechanizmy. Každý podnik si vytvára vlastné riešenie podľa individuálnych požiadaviek, čo znemožňuje univerzálny prístup a komplikuje štandardizáciu.

Architektúra mikroslužeb

Architektúra mikroslužieb predstavuje evolučné rozšírenie prístupu orientovaného na služby (SOA), ktoré kladie dôraz na ešte vyšší stupeň modularity a nezávislosti jednotlivých komponentov [35]. Rozdiel týchto dvoch architektúr je zobrazený na obrázku 3.2. Mikroslužby sú navrhované ako malé, samostatne nasaditeľné a nezávislé jednotky, pričom každá z nich realizuje konkrétnu obchodnú funkcionalitu. Na rozdiel od SOA, kde služby často zdieľajú databázy a infraštruktúru, mikroslužby majú vlastný kódový základ, databázu a komunikujú jednoduchými protokolmi (napr. HTTP), čo výrazne znižuje ich vzájomnú závislosť [7, 35].



Obr. 3.2: Schémy popisujúce architektúru mikroslužieb (vpravo) a SOA (vľavo). Na obrázku je vidieť hlavný rozdiel v modularite, kde architektúra mikroislužieb viac využíva rozdelenie do služieb ako SOA[4].

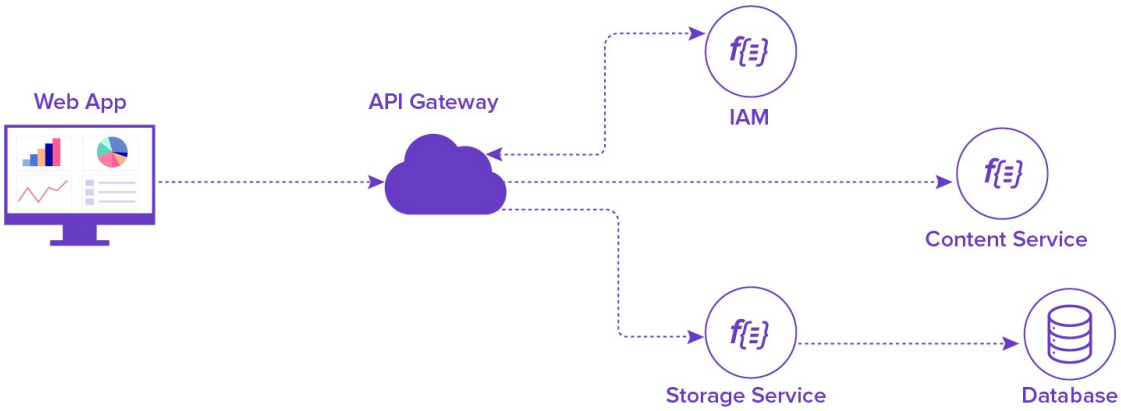
[4]Prevzaté z: 3pillarglobal.com/wp-content/uploads/2021/12/Microservices-vs-SOA-communicaiton.png

Výhodami aj nevýhodami architektúry sa zaoberá zdroj [59]. Pozitívnou charakteristikou je jednoduchá škálovateľnosť, vzhľadom na povahu tejto architektúry sa mikroslužby ľahšie škálujú ako monolitická architektúra. Jasné hranice odrážajú jasné zodpovednosti, a to ako z hľadiska služby (mikroslužba by mala vykonávať iba jednu vec) a vývojového tímu, kde jeden tím je jednoznačne zodpovedný práve za jednu mikroslužbu bez zdieľania zodpovednosti s inými tímami. Mikroslužby vedia byť nasadené nezávisle od seba a od zvyšku aplikácie, zatiaľ čo u monolitickej architektúry musia vykonávať nasadenie častí synchrónne.

Hlavnou nevýhodou podľa zdroja [35] je náročnosť na zdroje a kvalitu vzdelania vývojárov. Prevádzka viacerých nezávislých služieb si vyžaduje pokročilé praktiky procesov softvérového riadenia, sofistikované monitorovanie a orchestráciu systémov. Koordinácia testovania viacerých služieb si vyžaduje náročné integračné testy a testovanie kompletnej funkcionality aplikácie, čo môže predlžovať vývojový cyklus [59].

Bezserverová architektúra

Bezserverová architektúra, známa aj ako *Functions as a Service* (FaaS), predstavuje moderný a plne udalosťami riadený model, pri ktorom správu infraštruktúry zabezpečuje poskytovateľ cloudových služieb ako napríklad Google Cloud Functions[5] či IBM OpenWhisk[6] [2, 35]. Tento typ predstavuje softvérovú architektúru, v ktorej sú aplikácie rozdelené na „triggery“ (udalosti) a „akcie“ (funkcie). Tieto funkcie sú jednoduché, stavovo nezávislé a sú vykonávané na požiadanie prostredníctvom API, pričom samotný vývojár sa nemusí zaoberať ani správou infraštruktúry, ani hosťovaním [52]. Architektúra je znázornená na obrázku 3.3.



Obr. 3.3: Schéma zobrazujúce fungovanie bezserverovej architektúry[8]. Naznačená databáza v schéme predstavuje externú databázu.

Kľúčovým znakom serverless architektúry je, že zodpovednosť za dátové centrá, servery a prostredie, na ktorom je spúšťaná aplikácia, preberá cloudový poskytovateľ. Tento model tak kontrastuje s inými cloudovými prístupmi (napr. *Infrastructure as a service* (IaaS), *Platform as a service* (PaaS)), kde zostáva viac zodpovednosti na strane vývojára alebo prevádzkového tímu. V prípade FaaS sa teda znižuje technická záťaž spojená so správou a údržbou, čo umožňuje vývojárom plne sa sústrediť na obchodnú logiku aplikácií [52].

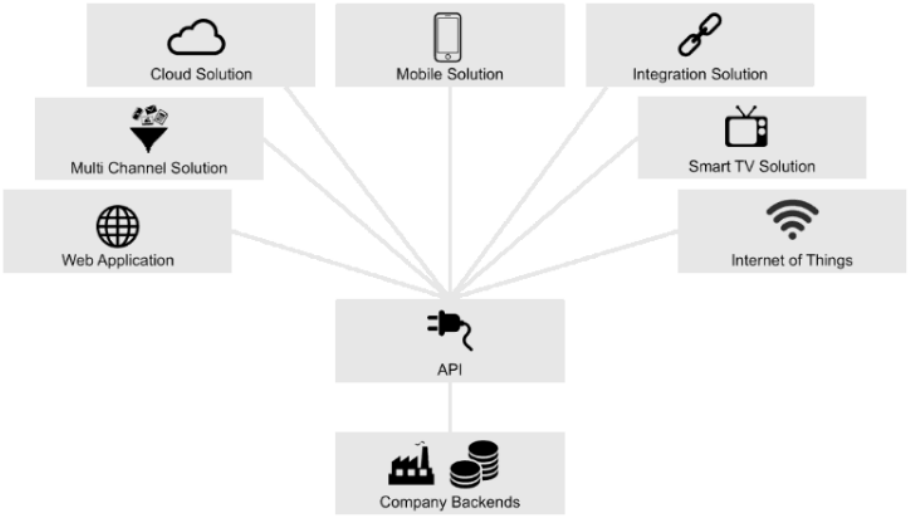
[5] cloud.google.com/functions
[6] developer.ibm.com/openwhisk/
[8] Prevzaté z: www.site24x7.com/blog/what-is-serverless-computing

Výhodami tejto architektúry je automatická škálovateľnosť, kde sa systém dokáže prispôsobiť aktuálnemu zaťaženiu. Vývojári sa môžu sústrediť výlučne na kód aplikačnej logiky, keďže niektoré ďalšie náležitosti sú v kompetencii poskytovateľa. Taktiež je táto architektúra efektívnejšia na náklady, čo je výhodné pre nepravidelné alebo občasné úlohy [2].

Závislosť na poskytovateľovi môže byť aj nevýhodou, riešenia sú často spojené s konkrétnou cloudovou platformou, čo komplikuje migráciu [35]. Môže vznikať aj nákladová neefektivita pri dlhšie trvajúcich procesoch. Bezserverový model architektúry je cenovo výhodný pri krátkych a nepravidelných úlohách, no náklady narastajú pri funkciách s vyšším výpočtovým výkonom alebo dlhším časom vykonávania. V takých prípadoch môže byť prevádzka nákladnejšia než pri tradičných modeloch [17]. Nevýhodou je tiež obmedzenosť pre veľké a komplexné aplikácie. Táto architektúra sa ukazuje ako efektívna najmä v menších organizáciách alebo pri jednoduchej funkcionalite. Pri rozsiahlych aplikáciách, ktoré vyžadujú koordináciu viacerých zložiek alebo zložitú logiku, môže implementácia bezserverového prístupu viesť k zložitosti, neprehľadnosti a nižšej výkonnosti systému [17].

3.2 Architektúra a návrh API

Aby bolo možné z užívateľského hľadiska pristupovať k tomu, čo ponúka daná aplikácia alebo iný softvér, je potrebné, aby existovalo isté rozhranie, vďaka ktorému vieme k aplikácii pristupovať. Pre prístup k týmto funkcionalitám je najčastejšie používané *Application Programming Interface* (API elementy), čo môžu byť napríklad triedy, metódy a polia poskytované návrhármi aplikácie či knižnice [47]. Priblíženie tohto konceptu je zachytené na obrázku 3.4.



Obr. 3.4: Základný koncept fungovania aplikačného rozhrania (API) [6].

Rovnako je koncept API využívaný pri vytváraní webových aplikácií, kde je využívané hlavne pre komunikáciu so serverom. Webové rozhrania API sa stávajú istým základom webových, mobilných a cloudových aplikácií spolu s aplikáciami so strojovým učením [60]. K webovej komunikácii vieme pristupovať jednoducho prostredníctvom URL. Webové API majú frekventovanejšiu dokumentáciu, ktorá človeku prirodzenejšie popisuje, ako dané URL

použiť pre rôzne zdroje servera. Pre bežného užívateľa je však potrebné tieto URL zaobaliť v podobe odkazov alebo tlačidiel, tvoriacich samotné užívateľské rozhranie webu. To tvorí jeden z problémov návrhu samotného webu [37, 47].

Každý typ dizajnu či návrhu si vyžaduje premyslené a informované rozhodnutia. Správne rozhodnutia sú kľúčové na vylepšovanie samotného produktu v istých ohľadoch, napríklad v dodávaní viac funkcionalít, lepšej kvality alebo lepšej užívateľskej prívetivosti. Lepšie kroky vykonané v návrhu typicky vedú k lepšiemu produktu a nie je tomu výnimkou ani pri návrhu samotného API. Podľa publikácie [6] je možné tieto rozhodnutia pri návrhu rozdeliť do štyroch skupín:

- **Architektonický návrh** – Pri návrhu API musia byť podstúpené kroky v súvislosti s problémami ako ktorý návrh či štýl použiť. Napríklad či použiť architektonický štýl *Representational State Transfer* (REST) alebo *Simple Object Access Protocol* (SOAP).
- **Návrh API na klientskej časti** – Keďže API tejto časti je najviac viditeľné pre koncového užívateľa, sú správne rozhodnutia v tomto prípade viac žiadúce.
- **Návrh API na serverovej časti** – Kroky vykonané v tomto návrhu týkajúce sa integrácie, transformácie, agregáciu, bezpečnosť a spracovanie chýb majú vplyv na samotnú funkčnosť API.
- **Ne-funkcionálny návrh** – V tomto prípade musíme podstúpiť rozhodnutia a kroky súvisiace s bezpečnosťou, výkonom, dostupnosťou a schopnosťou sa vyvíjať.

RESTful API

Representational State Transfer (REST) je architektúra používaná na navrhovanie služieb využiteľných na rôznych platformách a prostrediach na podporu interoperability a *World Wide Web* (WWW) [37]. REST sa stal široko používaným štandardizovaným spôsobom publikovania služieb naprieč internetom. Hlavné dve vlastnosti tejto architektúry sú bezstavovosť a pripravenosť na použitie naprieč rôznymi platformami [21]. Vychádza z tradičného klient-server modelu, v ktorom klienti komunikujú so servermi cez sieť. REST definuje jednotné rozhranie pre interakciu medzi systémovými komponentmi, založené na identifikácii zdrojov a dynamickom poskytovaní dát. Každý zdroj je prístupný pomocou jedinečného koncového bodu – konkrétnej URL adresy, cez ktorú možno získať alebo manipulovať so špecifickými údajmi [8].

RESTful API teda označuje API, ktoré dodržiava pravidlá REST štýlu a poskytuje jasne definované, zdrojovo orientované rozhranie pre komunikáciu medzi systémovými komponentmi. Každý tento koncový bod je konkrétna implementácia funkcionality podnikového procesu [21]. Ako bolo v úvode spomínané, k týmto bodom môžeme z užívateľského hľadiska jednoducho pristupovať prostredníctvom URL, ktoré reprezentuje adresu miesta na serveri. Tieto API sú teda dostupné skrze *Hypertext Transfer Protocol* (HTTP) používaním štandardných metód ako GET, POST, PUT alebo DELETE.

GraphQL

GraphQL je dotazovací jazyk určený na implementáciu webových služieb, ktorý bol pôvodne vyvinutý spoločnosťou Facebook ako odpoveď na nedostatky tradičných architektonických prístupov, predovšetkým REST [8]. Nie je viazaný na konkrétnu databázu alebo úložisko, ale funguje nad existujúcim kódom a dátami. V podstate má GraphQL len jeden koncový

bod, ktorým je HTTP požiadavka typu POST [36]. Klient jednoducho odošle jeden požiadavok na GraphQL server, v ktorom presne špecifikuje, aké dáta potrebuje. Server následne odpovie objektom vo formáte JSON, ktorý obsahuje požadované údaje [8].

Poskytuje deklaratívny dotazovací jazyk, ktorý umožňuje klientom presne špecifikovať, aké dáta potrebujú. Na rozdiel od imperatívnych jazykov, v ktorých sa špecifikuje, ako má výsledok získať, v GraphQL sa definuje, čo sa konkrétne potrebuje. O to, ako sa tieto dáta získajú, sa postará server [36].

Dáta modeluje ako graf, kde každý uzol predstavuje objekt a každé pole objektu môže odkazovať na iný objekt (čo vytvára hrany medzi uzlami). Tento graf je definovaný pomocou schémy, v ktorej sú objekty popísané pomocou typov. Každý má svoje polia, ktoré majú určený dátový typ a názov [8].

GraphQL podporujú viaceré populárne programovacie jazyky ako Python, C#, JavaScript a ďalšie. Pre správne fungovanie GraphQL aplikácie je potrebné definovať tri hlavné časti a to mutácie (na úpravu, vkladanie a mazanie dát), dotazy (na získavanie dát) a typy, ktoré opisujú štruktúru jednotlivých polí alebo dátových prvkov [36].

Čo sa týka rozdielov oproti REST, v GraphQL sú servisné dáta sprístupnené vo forme grafu, ktorý je reprezentovaný schémou. Naopak, v REST aplikáciách server implementuje zoznam koncových bodov. GraphQL taktiež definuje dotazovací jazyk, ktorý umožňuje klientom presne špecifikovať, aké polia požadujú zo servera. V REST službách sú dotazy definované prostredníctvom jednotlivých koncových bodov, kde každý vracia vopred určený súbor polí, ktoré reprezentujú dáta o určitom zdroji. Na druhej strane, v GraphQL odpoveď zodpovedá štruktúre dotazu [8].

3.3 ORM (Objektovo-Relačné Mapovanie)

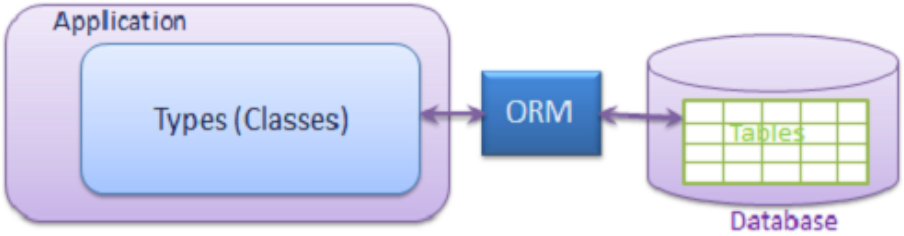
So serverovou časťou informačného systému súvisí tiež databáza, s ktorou komunikuje. Existujú rôzne spôsoby, ako k tejto komunikácii medzi týmito dvoma časťami pristupovať.

Bežne používaný systém relačnej databázy je už do veľkej miery známy a široko používaný pre ukladanie a načítanie dát. Objektovo orientované programovanie (OOP) sa používa ako hlavný nástroj na vývoj rozhraní relačných databáz na účely získavania a manipulácie s informáciami užívateľsky prívetivou formou [3]. Objektovo orientovaná aplikácia zvyčajne používa už spomínané rozhranie API, ktoré zase volá ovládač na komunikáciu s databázou [4]. Je samo o sebe náročné manipulovať a kontrolovať dáta uložené v relačnej databáze s použitím objektovo orientovaného programovania napríklad skrze interakciu s SQL [3]. Je možné prepojiť tieto dva aspekty a zefektívniť programovanie aplikačnej logiky a prístupu k databáze.

Efektívnym riešením podľa zdroja [48] je použitie *Object-Relational Mapping* (ORM). Tento framework bol vytvorený vo verejne dostupnom projekte pre programátorov v jazyku Java v roku 2002.

V sfére programátorov a vývojárov webových aplikácií je preferované zaobchádzať s perzistentnými dátami existujúcimi v nejakom programovom objekte, než zaobchádzať s čistými SQL príkazmi pre prístup k dátam databázy. ORM framework znižuje istý nesúlad medzi OOP a relačne orientovanou databázou prevodom údajov z relačnej databázy na vhodné programovacie objekty a naopak [48].

[10]Prevzaté z: www.code-sample.com/2016/01/orm-object-relation-mapper.html



Obr. 3.5: Schéma naznačujúca použitie ORM prístupu u všeobecnej aplikácie[10]. Tento prístup je možné aplikovaž aj pre webovú aplikáciu.

Môže sa zdať, že použitie ORM je v celku priamočiare, ako naznačuje aj obrázok 3.5. V skutočnosti sú objektovo-relačné mapovače pozoruhodne rôznorodé v rozsahu svojich prístupov k problému ORM a vo funkciách, ktoré poskytujú [4].

Mapovacie paradigma

Existuje množstvo perspektív, z ktorých možno pristupovať k ORM. Výber použitého vzoru (paradigma) je možno najzákladnejším rozhodnutím zvažovaným vo vývoji alebo výbere systému ORM [4]. Medzi základné prístupy, z ktorých sa vieme k ORM pristupovať, sú nasledujúce tri.

- **Metadáta-orientované**
  V prvom type ORM systému je jediným vstupom vývojára aplikácie do procesu ORM metadáta. Samotný ORM systém potom generuje kód. Tento prístup, nazývaný aj generovanie kódu. Tento prístup sa líši podľa potrieb konkrétnej aplikácie – ak sa systém vytvára od základov, je praktické, ak mapovač dokáže vytvoriť databázu. Pri použití existujúcej (*legacy*) databázy však mapovač databázu negeneruje, ale prispôsobí kód existujúcej schéme [55]. Výhodou je jednoduchosť pre vývojára od ktorého sú odľahčené isté mechanizmy ako ukladanie údajov, či štruktúra mapovaných objektov. Nevýhodou je istá strata flexibility. Vývojár je obmedzený na funkcionalitu poskytovanú mapovačom [4].

- **Aplikačne orientované**
  V tomto prístupe vývojár aplikácie píše objektovo orientovaný kód pre celú aplikáciu, vrátane objektov, ktoré sa majú mapovať (okrem ich perzistentného kódu). Mapovač, zvyčajne s pomocou dodatočných metadát od používateľa, generuje perzistentný kód pre tieto objekty [4]. Výhodou je oproti predošlému prístupu flexibilita. Vývojár môže do mapovaných tried zapisovať ľubovolnú logiku.

- **Databázovo orientované**
  V tomto prístupe je mapovanie orientované na databázu. Vývojár aplikácie vytvorí databázu, ktorú mapovač preskúma a odvodením z nej (často s pomocou metadát) určí štruktúru tried. Následne vygeneruje kód pre objekty. Zatiaľ čo mapovanie orientované na aplikáciu umožňuje vývojárom pracovať v známom objektovo-orientovanom prostredí, databázovo-orientované mapovanie pôsobí pre vývojárov menej intuitívne a obmedzuje ich možnosť upravovať mapované triedy. Tento model je však vhodný pre aplikácie, kde je dôležitá štruktúra dát a vzťahy medzi nimi, zatiaľ čo obchodná logika hrá menšiu rolu [4].

Ukladanie do vyrovnávacej pamäte

Mapovacie systémy môžu často dosahovať aj vyšší výkon ako ručne písaný kód. Kľúčovým bodom je zabudovanie vyrovnávacej pamäte (*cache*) [4]. Hlavným výkonnostným problémom je zvyčajne v databázovo orientovaných aplikáciách prístup do samotnej databázy.

Správa vyrovnávacej pamäte sa týka poskytovania a riadenia pamäťového priestoru na prístup k objektom v operačnej pamäti po ich načítaní z disku. Uložením do vyrovnávacej pamäte často používaných objektov alebo objektov s vysokým výskytom referencií sa zrýchľuje prístup k týmto objektom, pretože nie je potrebné ich opätovne načítať z disku [69]. Použitím vyrovnávacej pamäte na lokálne uchovávanie výsledkov dotazov je možné znížiť frekvenciu komunikácie s databázovým serverom.

Táto vyrovnávacia pamäť je súčasťou objektovo-relačne mapovaného systému a je oddelená od akejkoľvek vyrovnávacej pamäte spojenej so systémom správy databáz. Podľa [5] vieme klasifikovať ORM cache na základe ich rozsahu.

- **Cache pre transakciu**: Cache je spojená s konkrétnou „jednotkou práce“, ktorá môže predstavovať databázovú transakciu alebo významnú operáciu v aplikácii.
- **Cache pre proces**: Cache môže byť zdieľaná viacerými transakciami v rámci jedného procesu.
- **Cache pre klaster**: Cache môže byť zdieľaná medzi viacerými procesmi alebo viacerými strojmi. Tento rozsah vyžaduje vzdialenú komunikáciu medzi procesmi na zabezpečenie konzistencie a replikáciu dát na všetky uzly.

Čo sa týka samotnej implementácie napríklad v jazyku Python, je možné použiť framework *Odoo*, vďaka ktorému vieme použiť dekorátory funkcií. Cache v ORM systémoch využíva princíp „najmenej používané objekty“ (Least Recently Used, LRU), čo znamená, že kľúče, ktoré nie sú často používané, budú z cache odstránené. Nesprávne používanie cache však môže spôsobiť viac problémov než úžitku. Napríklad, ak metóda vždy prijíma rôzne argumenty, systém sa najskôr pokúsi vyhľadať výsledok v cache, ale ak tam nebude, metóda sa aj tak vykoná, čo môže zbytočne zaťažiť výkon [53].

3.4 Tvorba dokumentácie API

Dokumentácia API je technický dokument, ktorý poskytuje pokyny, ako efektívne používať a integrovať rozhranie. Slúži ako stručná príručka obsahujúca všetky potrebné informácie o práci s API, vrátane podrobností o funkciách, triedach, návratových typoch, argumentoch a ďalších aspektoch [63].

Interná API dokumentácia podporuje spoluprácu medzi tímami, znižuje duplicitu kódu a uľahčuje zaškolenie nových zamestnancov, čo zvyšuje efektivitu práce na spoločnom cieli. Na druhej strane, verejná API dokumentácia pomáha potenciálnym používateľom pochopiť a vyskúšať toto rozhranie, čo vedie k vyššiemu využívaniu a tým aj k zvýšeniu zisku [51]. Podľa [25] na základe výskumov môže dokumentácia API nadobúdať tieto podoby:

- obyčajný text – jednoduchý na čítanie pre používateľov, ale nespracovateľný strojmi,
- manuálne vytvorená dokumentácia podľa štandardu OpenAPI — strojovo čitateľná, ale náchylná na chyby a nákladná,

- automaticky generovaná dokumentácia OpenAPI – všeobecná a strojovo čitateľná, no ťažko pochopiteľná pre používateľov,
- interaktívna dokumentácia OpenAPI — ľahko spracovateľná počítačmi a používateľsky prístupnejšia, no stále môže byť náročná na pochopenie obsahu API.

Dokumentácia webového API je často základná, neprehľadná, neúplná alebo dokonca nepresná [1]. Väčšinou je písaná manuálne ako obyčajný text, čo znemožňuje automatické generovanie strojovo čitateľnej špecifikácie. To predstavuje výzvu pre používateľov, ktorí musia API detailne analyzovať, aby ho mohli správne pochopiť a používať, čo si vyžaduje veľké úsilie a čas [25].

Aby bolo možné aplikačné rozhranie a jej špecifikáciu strojovo spracovať, je potrebné toto webové API vedieť nejak opísať v strojom pochopiteľnej forme. OpenAPI špecifikuje spôsob, akým sa dajú opísať HTTP orientované API, ktoré sú typicky RESTful. Definícia OpenAPI je formovaná v súboroch typu YAML alebo JSON, kde sa dá opísať vstup a výstup koncových bodov. Môže taktiež zahŕňať informácie o tom, kde je API prevádzkované, aká je požadovaná autorizácia pre prístup do neho a ostatné detaily potrebné spotrebiteľmi a programátormi [50].

Napriek rozšírenosti OpenAPI existujú alternatívy, ktoré vytvárajú špecifikáciu daného API. Alternatívami sú napríklad použitie RAML[11], ktorý špecifikáciu API vo vlastnom type súboru s koncovkou `.raml`. Dovoľuje teda vytvorenie špecifikácie vo viac čitateľnom formáte. Ďalšou alternatívou je API Blueprint[12]. Obsahuje dokument ako obyčajný text, podobný typu `Markdown`, pre opísanie celého webového aplikačného rozhrania alebo jeho časti.

Popis tejto dokumentácie sa sprístupní v interaktívnej podobe ako webové rozhranie [25]. Často používaným je nástroj Swagger[13]. Alternatívou je napríklad použitie nástroja Postman[14], ktorý umožňuje testovanie a zdieľanie dokumentácie [51] alebo použitie Redoc[15].

Podľa časti so špecifikáciou zo zdroja [63] by dokumentácia koncových bodov mala obsahovať použitú HTTP metódu, URL koncového bodu, odpoveď koncového bodu a jeho formát, vstupné parametre a popis koncového bodu.

[11] raml.org
[12] apiblueprint.org
[13] swagger.io
[14] www.postman.com/api-documentation-tool/
[15] redocly.com

Kapitola 4

Analýza existujúceho riešenia

Táto kapitola sa zaoberá samotnou analýzou a popisom doterajšieho riešenia webového informačného systému pre zobrazovanie a analýzu dát hlasovania zastupiteľstiev v Českej republike s názvom Zastupko.cz[1] (obrázok 4.1). Informačný systém okrem samotných hlasovaní zobrazuje jednotlivé zasadania zastupiteľstva, politické strany, členov politických strán a ich dochádzku spolu s niektorými analýzami týchto dát.

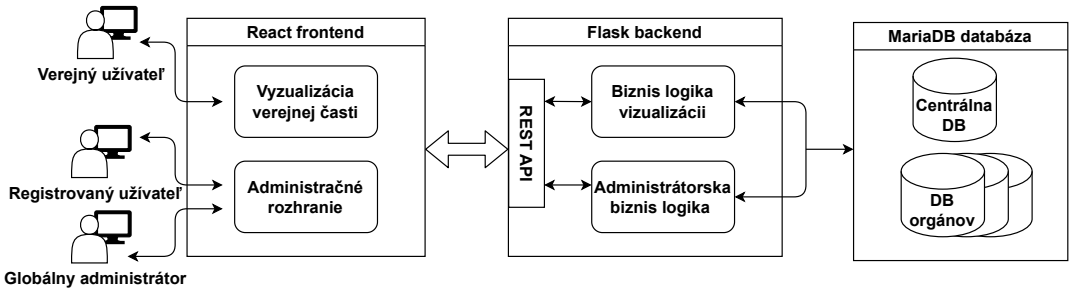
Tento informačný systém pôvodne pracoval s dátami pochádzajúcimi zo Zastupiteľstva mesta Brno [68]. Ďalej bol rozšírený o možnosť obsahovať záznamy z viacerých orgánov a miest [33]. Taktiež bol v spomínanej práci systém rozšírený o možnosť spravovať tieto záznamy prostredníctvom administrátorského užívateľského rozhrania. Riešenie bolo analyzované zo stavu v mesiaci október roku 2024.

Obr. 4.1: Ukážka užívateľského rozhrania a zobrazovaných dát v úvodnej stránke projektu Zastupko.cz.

[1] zastupko.cz

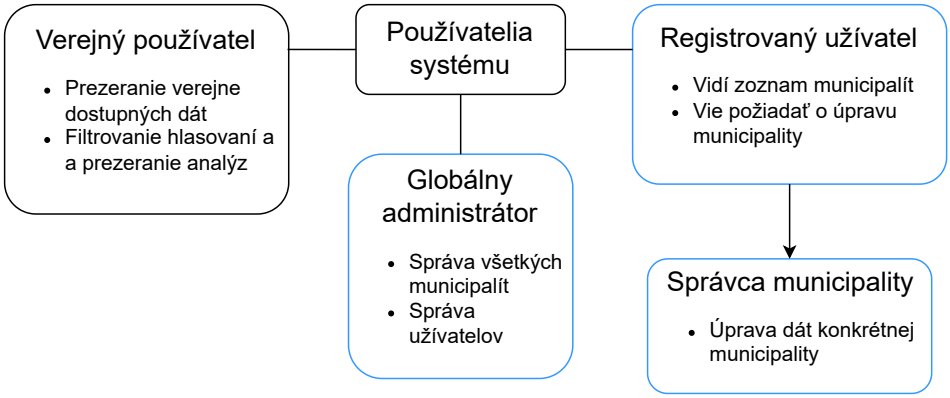
4.1 Architektúra systému

Web samotný je implementovaný ako webová aplikácia s trojvrstvovou architektúrou klient-server s použitím technológií zobrazených na obrázku 4.2. Systém sa rozdeľuje na dve časti a to **administračnú** pre správu dát a **verejnú** časť, ktorá je hlavným zámerom tejto práce. Preto budú časti tejto kapitoly týkajúce sa verejnej časti popísané podrobnejšie.



Obr. 4.2: Schéma architektúry a použitých technológii súčastného systému, naznačené sú hlavní časti a použité technológie. Klientská časť implementovaná v jazyku JavaScript s knižnicou **React** a serverová časť v jazyku Python s použitím frameworku **Flask**. Pre databázu bola použitá **MariaDB** (autory schémy Radoslav Kodaj a Štěpán Krejčíř [34]).

Užívateľské účty sa nachádzajú v administratívnej časti, kde je možné aj spravovať dáta. Systém taktiež dovoľuje vytvorenie užívateľa pre správu iba jednej konkrétnej municipality (ďalej zobrazené na obrázku 4.3). Účty bežného užívateľa systém neobsahuje, takže slúži primárne na voľné prezeranie dát kýmkoľvek, kto pristupuje na web. Primárne sa teda web zameriava na občanov Českej republiky s volebným právom. Príkladmi ďalších cieľových užívateľov môžu byť napríklad novinári, zamestnanci úradov a samotní zastupitelia.



Obr. 4.3: Schéma popisujúce užívateľov informačného systému. Registrovaný užívateľ po schválení žiadosti globálnym administrátorom prejde do stavu kedy je správca samotnej municipality. Modrou sú zvýraznení užívatelia, ktorý pre prácu so systémom potrebujú vlastný účet.

[1]Prevzaté z: swagger.io/tools/swagger-ui

Klientská časť

Pre orientáciu a smerovanie v aplikácii slúži knižnica `React Router`, použitá v jednom z hlavných súborov, kde sú nastavené cesty k jednotlivým stránkam, pomocou nej je definované mapovanie URL medzi stránkami. Komunikácia so serverom je realizovaná prostredníctvom knižnice `axios` a objektu `Promise`, vďaka ktorému je možné požiadavky realizovať na server asynchronne [19]. Volania rozhrania servera pomocou knižnice `axios` sú zoskupené na jednom mieste, ale ojedinele sa volania vyskytujú aj v jednotlivých dielčích implementáciách stránok. Ďalšia dôležitá knižnica je `PrimeReact`, využitá pre tvorbu komponentov v užívateľskom rozhraní ako napríklad tabuľky či rozbaľovací zoznam *dropdown*. Doplňujúci štýl HTML komponentov je doplnený v `CSS` súboroch.

Počiatočná stránka aplikácie je tvorená zoznamom municipalít miest a krajov, pri čom jednotlivé položky vedú na stránky zastupiteľstiev jednotlivých municipalít. V záhlaví každej stránky sa nachádza menu s odkazmi na vybrané stránky a v päte stránky informácie ako kontakt či odkazy na projekty. Vnútorný obsah stránok je daný podľa zvolenej municipality a zvolených dát. V implementácii klientskej časti sú jednotlivé stránky vytvorené formou komponentov založených na funkciách, kde v danej funkcii je obsah stránky v návratovej hodnote.

Každá municipalita má okrem úvodnej ďalšie 4 základné stránky o subjektoch, zastupiteľoch, zasadaniach a hlasovaniach zastupiteľov. Ďalšími stránkami sú jednotlivé analýzy pre zastupiteľstvá.

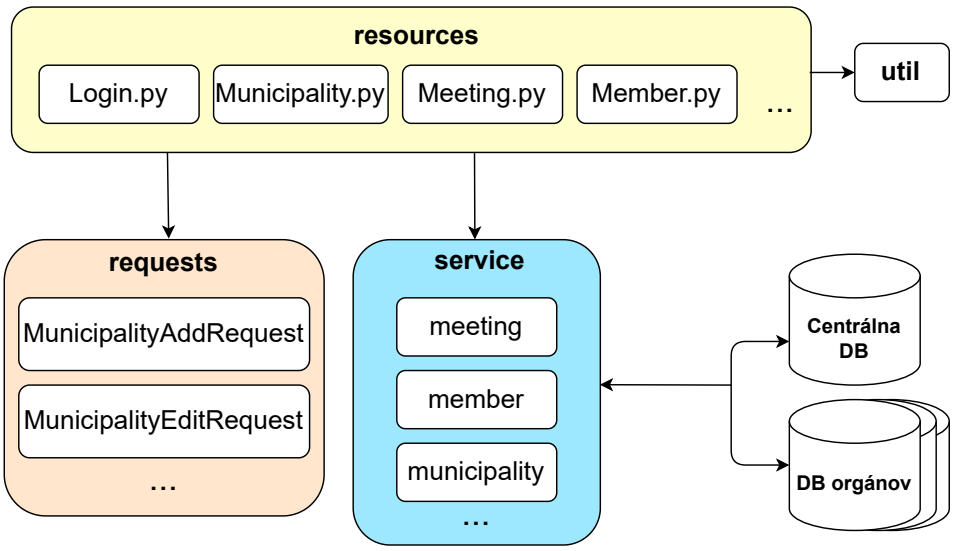
Užívateľské rozhranie administratívnej časti po prihlásení umožňuje zobraziť jednotlivé municipality a ich orgány. V jednotlivých orgánoch je možné zvoliť konkrétne funkčné obdobie a v rámci neho upravovať a pridávať nové dáta. Hlavný administrátor okrem toho vie pristupovať aj k zoznamu užívateľov a upravovať ich privilégia a spravovať žiadosti o prístup.

Serverová časť

Pre vytvorenie REST API bolo použité `Flask-Restful`, ktoré je rozšírením samotného frameworku Flask. Pomocou tohto rozšírenia boli vytvorené jednotlivé koncové body, ktoré sú reprezentované triedami. Získavanie dát z databázy je implementované dvomi spôsobmi rôzne používanými. Prvý spôsob používa knižnicu `mariadb` pre spojenie s relačnou databázou, kde pre dotazovanie je použitý čistý `SQL` kód. Druhý spôsob je implementovaný prostredníctvom knižnice `SQLAlchemy` a využitia ORM prístupu. V analyzovanom riešení je vytvorená špeciálna trieda, ktorá dedí z triedy `Api` z knižnice `Flask-RESTX` s názvom `ExtendedApi`. Jej úlohou je hlavne registrovať chybové hlásenia serveru.

Serverová časť sa okrem spoločných štartovacích súborov delí na dve časti a to **administratívnu** a **verejnú**. Koncové body obidvoch častí sa začínajú prefixom `/flask`. Administratívna časť má vlastné koncové body obsahujúce prefix `/flask/admin` ako napríklad koncový bod pre získanie všetkých municipalít `/flask/admin/municipality`. Jednotlivé triedy koncových bodov obsahujú implementáciu REST API metód najmä `PUT`, `DELETE`, `GET` a `POST`. Spojenie s databázou je realizované použitím vytvorenej triedy `ORMHandler`. Pre implementáciu tohto spôsobu získavania dát sú v administratívnej časti vytvorené triedy, ktoré reprezentujú jednotlivé objekty databázy a ich atribúty.

Administratívna časť oproti verejnej už obsahuje istú štruktúru zobrazenú na obrázku 4.4, kde je možné vidieť istý spôsob modularity. Zvlášť sú oddelené časti, ktoré implementujú aplikačnú logiku a operácie nad databázou. V tretej časti sú oddelené triedy využívané pri požiadavkách na databázu typu `PUT` a `POST`.



Obr. 4.4: Administratívna časť sa dá rozdeliť na tri logické časti a to časť ktorý zabezpečuje aplikačnú logiku (žltá farba), časť ktorá implementuje operácie nad databázou (modrá farba) a časť s implementáciou schém potrebných pri požiadavkách na databázu (oranžová farba).

Verejná časť serverovej strany má koncové body a triedy podobného štýlu obsahujúce iba spoločný počiatočný prefix `flask`. Keďže v tejto verejnej časti postačí čítať dáta z databázy, v triedach, ktoré implementujú koncové body, je použitá iba REST metóda `GET` pre získavanie dát z relačnej databázy. Táto časť servera využíva druhý spôsob spojenia s databázou, kde sú použité parametrizované `SQL` dotazy a databázový kurzor.

Triedy implementujúce koncové body API obsahujú logiku získavania dát z databázy a rovnako aj aplikačnú logiku. Všetky triedy sú rozmiestnené do súborov na jednom mieste a nie sú spojené po logických častiach. Tieto dve časti je vhodné rozdeliť pre väčšiu prehľadnosť a vniesť jednotnú štruktúru, ako je tomu u administračnej časti.

Verejná časť aktuálne obsahuje 25 koncových bodov a ich tried, ktoré ich implementujú. Body, ktoré sú priamo spojené s konkrétnym orgánom a funkčným obdobím, obsahujú dynamické segmenty, ktoré značia nasledujúce:

- `<municipality_id>` – názov municipality (napr. `brno`),
- `<body>` – orgán municipality ako rada alebo zastupitelstvo,
- `<council_id>` – identifikátor funkčného obdobia.

Nie však nejakým spôsobom stanovené presné usporiadanie týchto koncových bodov alebo ich rozdelenie do menných priestorov. Je možné ich však podľa názvov a zamerania rozdeliť do skupín, ako je to popísané v prílohe A, kde sú vypísané aj koncové body pôvodného riešenia.

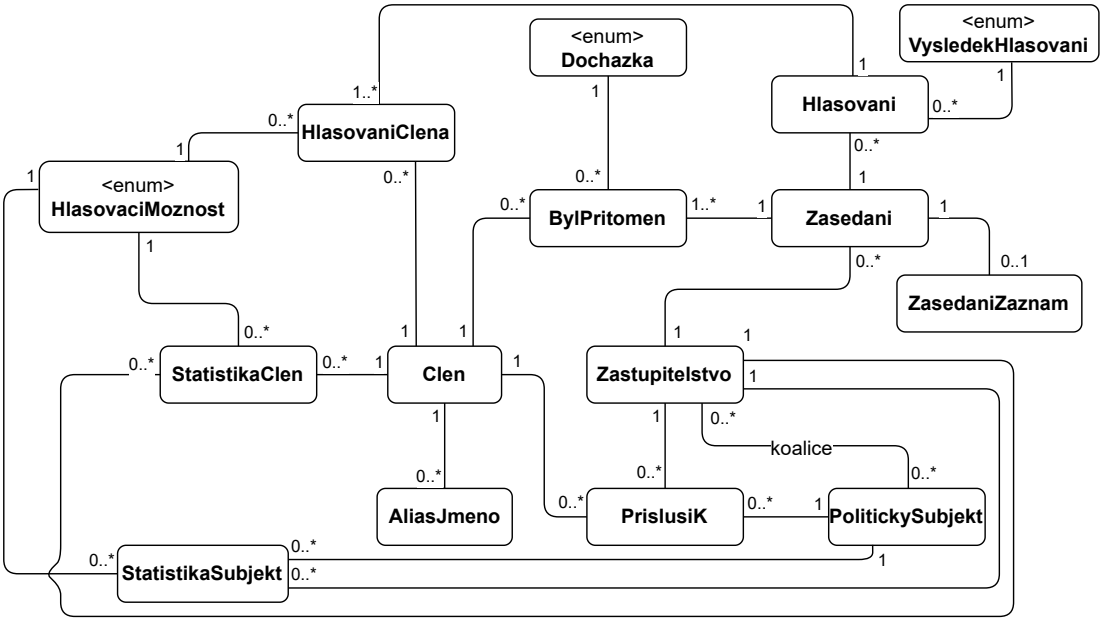
Koncový bod `/votes` implementovaný v triede `VotesQuery` je jediný, ktorý prijíma voliteľné argumenty využívané pre filtrovanie hlasovaní podľa zadaných parametrov. Filtrovať hlasovania je možné na základe popisu predmetu hlasovania, identifikátora užívateľa a jeho

konkrétnej voľby, výsledku hlasovania a čísla zasadania. Filtrovať je ďalej možné podľa vlastnosti hlasovania, ktoré je prenášané v parametre `show` a to podľa toho, či je hlasovanie tajné, validné alebo procedurálne.

Zvláštnymi koncovými bodmi sú tie, ktoré boli vytvorené špeciálne pre potreby klientskej strany. Okrem základného prefixu obsahujú v URL reťazec `/FE`. Sú využívané primárne pre vytvorenie zoznamov využívaných v komponente *dropdown*. Vracajú zoznamy zastupiteľov, zasadaní, výsledkov hlasovaní a politických strán.

Medzi základné entity databázy, ktorými táto časť systému pracuje, patria zastupiteľstvo, politický subjekt, hlasovanie, člen, zasadanie a hlasovanie člena. Okrem týchto základných entít obsahuje databáza aj ďalšie zobrazené v schéme 4.5 spolu so vzťahmi medzi nimi. Každá entita je reprezentovaná vlastnou tabuľkou v databáze. Existujú taktiež tabuľky reprezentujúce číselníky, napríklad pre výsledky hlasovania.

Potrebnou entitou pri členoch je tabuľka `AliasJmeno`. Jej úlohou je držať záznamy o zmenách mien (aliasov) členov zastupiteľstva. Záznam okrem samotného aliasu mena nesie aj záznam, dokedy člen dané meno používal. Entita je dôležitá napríklad z hľadiska zmeny priezviska v prípade manželstva.



Obr. 4.5: Diagram entít v databáze systému pre vizualizáciu hlasovania zastupiteľstiev Českej republiky. Pre jednoduchosť nie sú uvedené všetky väzby medzi entitami, nepoužívané tabulky a atribúty. Pomocou `<enum>` sú vyznačené číselníky.

Ďalšou významnou entitou je `PrislusiK`, ktorá reprezentuje členstvo člena v politickej strane pre určité obdobie. Časový interval príslušnosti k politickému subjektu sa určuje podľa dátumu zasadania, na ktorom sa člen zúčastnil. V databáze existujú záznamy o prítomnosti člena (`BylPritomen`) a jeho hlas (`HlasovaniClena`) na každom zasadnutí.

V databáze sa nachádzajú tabuľky, ktoré plnia inú hodnotu než len reprezentovanie entity. Zvláštnymi tabuľkami, ktoré sú obsiahnuté v databáze, sú:

- `BylPritomen`,

- `StatistikaClen`,
- `StatistiaSubjekt`.

Ich hlavným účelom je sumarizovať dáta k dochádzke, počtu hlasovacích možností za stranu a člena. Slúžia hlavne teda na isté urýchlenie výpočtu, ktorý by bol inak zdĺhavý a pracný. Svojím spôsobom predstavujú istú dočasnú pamäť s tým rozdielom, že sa dáta v nich pri manipulácii s dátami aktualizujú.

Serverová časť okrem samotných funkčných súborov obsahuje taktiež časť obsahujúcu migrácie databázy. Tieto migrácie slúžia pre hromadné úpravy tabuliek databáz, ktoré vznikli počas ďalšieho vývoja a ktorými bolo potrebné zmeniť pôvodnú štruktúru tabuliek databázy.

Testy serverovej časti

V súčasnej implementácii existujú aj testy serverovej časti a jeho rozhrania. Implementácia je presunutá mimo časť so samotnou serverovou implementáciou. Testy serverového API sú implementované s použitím testovacieho rámca Pythonu s názvom `Pytest`.

Vykonávaných testov je dokopy 141 rozmiestnených do 13 testovacích súborov, kde sa kontrolujú výstupné dáta a návratový kód servera.

Najskôr prebieha overenie nastavenia prostredia a pripojenia na databázu. Testované sú obidve časti, aj administračná, aj verejná. Testy z podstatne väčšej časti testujú administračnú časť oproti verejnej. V jej prípade je dôležitý testovací súbor, ktorý sa zameriava iba na čítanie z databázy. V tomto prípade sa porovnávajú dva súbory typu `JSON`. Jeden, ktorý vráti server, a druhý s požadovanými dátami.

Dáta pre testy pochádzajú z testovacích databáz oddelené zvlášť od databáz používaných pre vývoj s predponou `testovaci_`. Pre tento účel je preddefinovaný `SQL` skript pre manuálne vloženie testovacích dát mesta Brna a testovacích dát centrálnej databázy. Ďalej pre testovanie metód vkladania, úpravy a mazania dát je potrebné vytvoriť manuálne dodatočné prázdne databázy, ktoré sa po spustení testov automaticky vyčistia. Pri vkladaní sa vložia a vytvoria tabuľky automaticky testami na základe spomínaného skriptu.

4.2 Nedostatky aktuálneho riešenia

V tejto sekcii sú zanalyzované problémy a požiadavky na verejnú stránku serverovej časti. Niektoré nedostatky boli už opísané vyššie, v tejto sekcii budú rozobrané podrobnejšie s bližším kontextom k analyzovanému riešeniu informačného systému.

1. Nejednotný prístup do databázy

Jedným z prvotných problémov je istá nekonzistentnosť k prístupu do databázy. Ako už bolo spomínané, v systéme môžeme pozorovať dva prístupy, kde jeden používa parametrizované `SQL` a druhý používa princíp ORM. Pôvodné riešenie pracovalo s využitím čistých `SQL` požiadavkov v podobe reťazca. Ďalšie rozšírenie systému zaviedlo čiastočné používanie ORM. Bolo by taktiež vhodné, aby systém používal jednotný spôsob volania databázy namiesto dvoch oddelených spôsobov.

2. Staršia verzia knižnice

Popri analýze kódu bolo zistené používanie staršej verzie dôležitej knižnice `SQLAlchemy`. V niektorých prípadoch môže byť kľúčové zaobstarať, aby systém používal najnovšie verzie knižníc. Tento aspekt môže byť dôležitý z hľadiska bezpečnosti. Je preto vhodné kompaktnosť novších verzií preskúmať a prispôsobiť patrične zdrojový kód.

3. Tvorba aliasov a nepoužívané tabuľky databázy

V systéme bol implementovaný spôsob, ktorý rieši problém, kedy má člen zastupiteľstva zmenené meno (najčastejšie priezvisko) a systém potrebuje držať záznamy o hlasovaní tohto člena za obdobia v obidvoch (alebo viacerých) prípadoch verzie mena. Po analýze kódu a databázy bolo zistené, že pre záznamy týchto aliasov sú uložené iba v entite `AliasJmeno`. V časti s migráciami je prítomný skript, ktorý dáta zo starej tabuľky `Alias` presúva do novej, nebola už však vykonaná analýza, či je možné používanie danej entity z kódu odstrániť. Databázy obsahujú ďalšie tabuľky v databázach orgánov, ktoré v samotnej aplikačnej logike nie sú používané, a to `KlicoveSlovo`, `HlasovaniObsahujeSlovo`, `Oblast`, `PredvolebniSlib`, `Slibil`, `SlibilObsahujeSlovo` a `Zdroj`. Tieto nepoužívané tabuľky databáz je vhodné odstrániť nielen z funkčnej databázy, ale taktiež z testovacieho prostredia, ktoré tieto entity obsahuje, ale nijak netestuje.

4. Presun niektorej logiky z klientskej časti

Ďalším problémom je výskyt niektorých výpočtov na stránkach analýzy nachádzajúcich sa na klientskej časti. Strana, ktorá implementuje užívateľské rozhranie, by sa mala primárne zameriavať na spracovanie dát od servera a ich zobrazenie užívateľovi v prívetivej forme. Ostatný výpočet je lepšie presunúť na serverovú časť aj z hľadiska rýchlejšieho výpočtu. Jedným z týchto aspektov je počítanie percentuálnej dochádzky členov na zasadaniach a hlasovaniach. Ďalší je v časti, ktorá sa zameriava na porovnávanie dát zastupiteľov. Výpočet časového úseku, za ktorý sú zastupitelia porovnávaní, taktiež prebieha na klientskej strane a mal by byť tiež vypočítaný na serverovej časti.

5. Špecifikácia API

V podkapitole 3.4 bola okrem technických náležitostí vysvetlená dôležitosť dokumentovania API. Súčasné riešenie neobsahuje generovanie špecifikácie aplikačného rozhrania a interaktívnu dokumentáciu API ako napríklad Swagger alebo Postman. Pre ďalší vývoj API a lepšiu testovateľnosť je vhodné tento aspekt integrovať do súčasného API.

6. České kľúče v odpovediach koncových bodov

Pri analýze riešenia je vidieť istú inkonzistenciu medzi českým a anglickým jazykom. Keďže sa pôvodný zámer iba na české zastupiteľstvá zmenil, je nutné preložiť verejné API do anglického jazyka. Zaviesť anglické kľúče v posielanom formáte `JSON` je vyžadujúce aj z hľadiska istého nepísaného štandardu vo webových aplikáciách. V plánovanom zavedení interaktívnej dokumentácie API je taktiež vhodné, aby obsahovala anglické kľúče, ktorým by mohla porozumieť aj zahraničná verejnosť.

7. Zmeny pre PSP

Paralelne vznikajúca bakalárska práca sa sústreďuje na dáta z orgánov na národnej úrovni, kde sa berie do úvahy aj Poslanecká snemovňa Parlamentu (PSP) a Senát. Jedným z diskutovaných rozšírení je podpora viacdenných zasadnutí zastupiteľstva, ktoré sa vyskytujú aj v prípade orgánov municipalít aj PSP. Súčasný systém hlasovania zastupiteľstiev viacdenné zasadania nepodporuje a je vhodné tento prípad doplniť aj v rámci časti, ktorou sa zaoberá táto práca. Úpravy pre PSP taktiež zahŕňajú vytvorenie entít pre správu dokumentov. Zahŕňajú tiež zaznamenávanie vedenia zastupiteľstva v danom orgáne. Súčasná implementácia vedenia orgánu zaznamenáva iba prostredníctvom vzťahu medzi entitami `Zastupitelstvo` a `Clen`. Túto väzbu je však praktickejšie reprezentovať v novej entite, ktorá by zaznamenávala pozície lídrov a ich funkcie. Tieto informácie už z časti zahŕňa entita `Zastupitelstvo`, takže je nutné tieto dáta presunúť migráciou do novej entity.

8. Rozdelenie dát podla členstva v strane

Zastupitelia počas jedného funkčného obdobia môžu meniť členstvo v politických stranách, ku ktorému prislúchajú. Informačný systém tieto prípady zohľadňuje a zobrazuje obdobia, za ktoré bol zastupiteľ prítomný v inej strane. Avšak už nerozdeľuje a nezobrazuje dáta na základe obdobia a príslušnosti k politickej strane. Užívateľ teda vidí dáta z hlasovaní za celé funkčné obdobie a musí hľadať, ktoré hlasovania boli vykonané za akú stranu. Tento problém nastáva v triede koncového bodu, ktorý rieši detail zastupiteľa a tiež v časti, ktorá rieši analýzu dochádzok všetkých zastupiteľov. Pri celkovej analýze sa môže zdať, že danú dochádzku mal člen iba v danej strane, ale v skutočnosti za danú stranu mal dochádzku inú ako v predošlej.

9. Nepresné percentuálne zobrazenie

So spomínanou analýzou dochádzky súvisí aj zobrazenie jej hodnôt. V tomto prípade sa dochádzka počíta na serverovej strane. Hodnoty dochádzky sú vyjadrené v percentách a zaokrúhlené na celé čísla. Po analýze riešenia bolo zistené, že zaokrúhľovanie na celé čísla spôsobuje napríklad to, že ak mal zastupiteľ dochádzku 99.8%, je táto hodnota zaokrúhlená na 100%, čo môže spôsobovať nepresnú interpretáciu dát.

Ďalšie analyzované problémy tiež súvisia s prehľadnosťou kódu a lepšou čitateľnosťou. Vhodné je tiež pridanie komentárov pre lepšie dokumentovaný a prehľadnejší kód systému.

Kapitola 5

Návrh

Z predošlej analýzy architektúry systému sú v tejto kapitole navrhnuté zmeny pre jej vylepšenie a modularitu, čo umožní praktickejšiu rozšíriteľnosť verejného modulu systému. Z analyzovaných problémov sú ďalej navrhnuté úpravy, ktoré je potrebné vykonať pre optimálnejšiu funkčnosť niektorých častí systému. Navrhnuté sú tiež zmeny potrebné pri ďalšom vývoji klientskej časti a vzhľadom na vznikajúce požiadavky na tento systém.

5.1 Zmeny architektúry serverovej časti

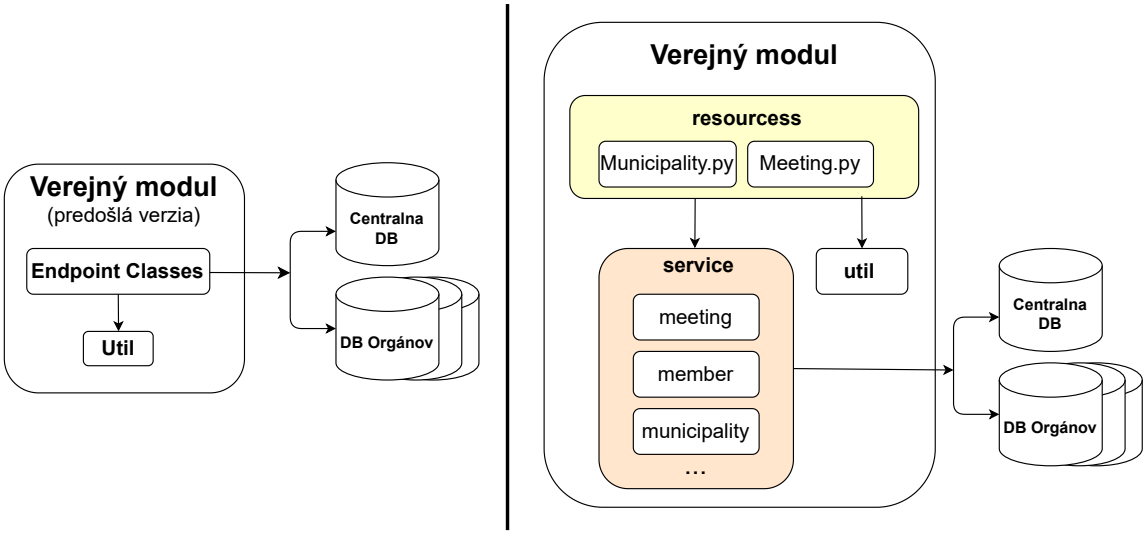
Celková architektúra systému zostáva nepozmenená tak, ako bola zobrazená na obrázku 4.2, kde klientská časť komunikuje so serverovou časťou prostredníctvom REST API. Zmeny sa budú týkať predovšetkým aplikačnej logiky, súborovej štruktúry a zmien vo využívaní zdrojov vo verejnej časti systému. Architektúra administratívnej časti bude ponechaná nezmenená.

Súčasný stav architektúry súborov, ktoré implementujú verejnú časť rozhrania serveru, je veľmi jednoduchý a nie je rozdelený na logické celky. Inšpiráciu a námet je možné nájsť v administračnom module (obrázok 4.4). Po novom budú časti, ktoré implementujú triedy pre koncové body verejnej časti, presunuté do vlastnej zložky. Každá trieda je implementovaná vo svojom vlastnom súbore spolu s volaniami databázy. V novej architektúre by tieto časti boli rozdelené do troch menších skupín (obrázok 5.1). Od administračnej časti sa líši hlavne častou `Requests`, ktorá vo verejnom module nie je potrebná, nakoľko v ňom dáta nie sú zapisované do databázy, iba z nej čítané:

- **Získavanie dát**
  Do tejto časti sa oddelia volania databázy a získavania dát. Keďže je potrebné volať databázu s rôznymi kombináciami dát, preto budú volania databázy rozdelené logicky do tried, ktoré súvisia s danou databázou. Vďaka tomu môžeme dosiahnuť využitie volaní databázy, ktoré sú podobné ale predtým boli napríklad na dvoch miestach.

- **Triedy koncových bodov**
  V týchto častiach bude obsiahnutá pôvodná aplikačná logika a spracovanie dát pre vytvorenie `JSON` súboru, ktorý je posielaný na klientskú stranu. Triedy budú používať volania databázy skrze metódy tried z predošlého bodu. Tým sa zjednoduší kód a primárne by sa sústredilo na aplikačnú logiku v kóde týchto súborov.

- **Pomocné súbory**
  Tu sa budú nachádzať súbory s definovanými pomocnými funkciami, ktoré sa často opakujú v triedach koncových bodov ako napríklad pre zistenie aliasov zastupiteľov.



Obr. 5.1: Porovnanie starej (vľavo) a navrhovanej (vpravo) schémy architektúry verejnej časti serverovej strany. Farebné časti predstavujú analogické využitie ako v prípade schémy na obrázku 4.4.

Ako bolo spomínané v kapitole 4, súčasné riešenie využíva dva spôsoby pre posielanie požiadavkov na databázou. Pre zabezpečenie optimálnejšej konzistencie v novej architektúre sa zjednotí táto časť so spôsobom, aký využíva administratívna časť. Zamedzilo by sa tak používaniu čistého `SQL` posielaného na databázu v podobe reťazcov a jednotlivé parametre by sa vedeli ľahšie vkladať do vytváraného požiadavku.

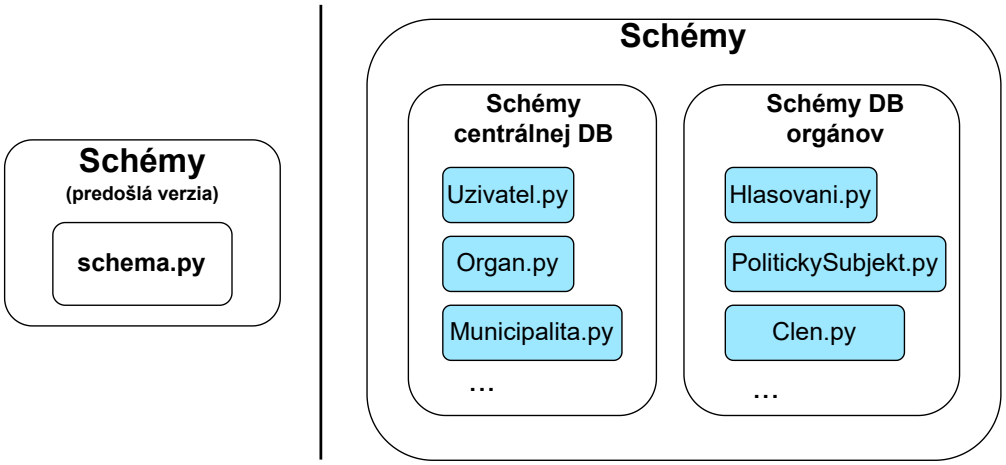
Pre vytvorenie požiadaviek za použitia ORM bude potrebné využiť triedy implementujúce schémy tabuliek databázy. Tieto schémy budú po novom rozdelené do logických celkov podľa súvislostí. Napríklad tabuľky spojené s hlasovaním ako:

- `Hlasovani`,
- `HlasovaciMoznost`,
- `HlasovaniClena`.

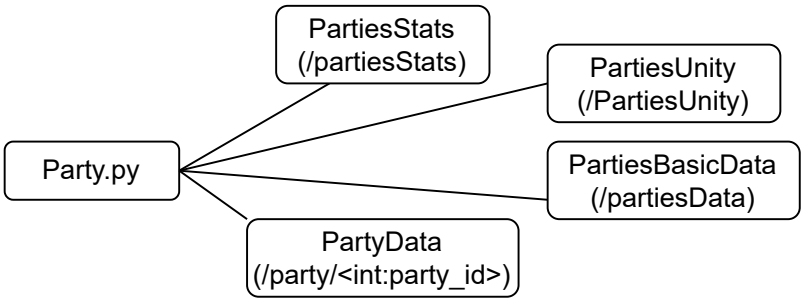
Boli by obsiahnuté v zvlášť súbore od ostatných. Všetky tieto triedy budú rozdelené do dvoch skupín podľa toho, či implementujú entity pre centrálnu databázu alebo databázu municipality, ako je to zobrazené na obrázku 5.2.

Podobný princíp spojenia súborov je vhodný aj v prípade tried koncových bodov. Keďže sa rozsiahle volania databázy presunú do oddelenej zložky, zostáva priestor na spojenie týchto tried, ktoré spolu súvisia, do jedného súboru ako v prípade zobrazenom na obrázku 5.3. Bude preto potrebné upraviť import týchto tried pri definícii API, čo však tiež spôsobí ich zjednodušenie.

Novo vzniknutým koncovým bodom podľa požiadaviek, ktoré sa vyskytli počas návrhu, je koncový bod, ktorý implementuje vyhľadávanie v danej municipalite a funkčnom období. Vyhľadávanie bude prebiehať na základe reťazca v parametri `search`. Vďaka tomu



Obr. 5.2: Schéma popisujúce rozdelenie schém entít databáz v porovnaní s predošlou verziou (vľavo) a navrhovanou (vpravo). Mená súborov sú vytvorené podla samotných názvov entít ktoré reprezentujú.



Obr. 5.3: Schéma zobrazujúce zjednotenie tried koncových bodov do jedného súboru používaných pri získavaní dát politických strán.

bude jednoduchšie nájsť pre užívateľa požadované dáta. Nový koncový bod bude posielať odpoveď, v ktorej sa budú nachádzať výsledky z vyhľadávania z tabuliek:

- `Clen` – vyhľadávanie členov podla mena a priezviska,
- `Hlasovani` – vyhľadávanie hlasovaní podla toho či atribút `predmet` obsahuje daný podreťazec,
- `PolitickySubjekt` – vyhľadanie politických subjektov podla názvu a skratky subjektu.d

Do novej architektúry by bolo presunuté aj testy API a migrácie databázy, ktoré by bolo prehľadnejšie umiestniť spolu s celkovou implementáciou servera a API. Keďže sú administrátorská a verejná časť zvlášť oddelené, zavedením podobnej architektúry vo verejnej časti by sa zachovala konzistencia medzi týmito dvoma modulmi. Návrh rozmiestnenia súborov v adresári je možné vidieť v prílohe B.

Štruktúra novej architektúry vychádza zo schémy 4.4, primárny rozdiel vo verejnej časti je v zložke `requests/`, ktorá v navrhovanej architektúre verejnej časti chýba, pretože táto

časť neimplementuje požiadavky typu `POST`, `PUT` a `DELETE`. V administrátorskej časti je zložka `requests/` miestom, kde sa nachádzajú triedy implementujúce tieto typy požiadaviek na databázu. Je vidieť, že tento spôsob štruktúry vie byť univerzálny naprieč rôznymi systémami, ktoré implementujú REST API. Štruktúra môže byť jednoducho ďalej rozšíriteľná o potrebné moduly. To, že štruktúra vychádza z administračnej časti, zabezpečí aj jednotnosť a konzistenciu týchto dvoch modulov, v ktorých sa neskôr bude jednoduchšie zorientovať.

5.2 Dokumentácia aplikačného rozhrania

Ďalším dôležitým spomínaným problémom je chýbajúca špecifikácia API opísaná aj v analýze problémov pod bodom č. 5. Ako už bolo spomenuté, podľa sekcie 3.4 prístupov ku generovaniu API dokumentácie sú rôzne. Jedným z vhodných riešení je vedieť špecifikáciu generovať priamo z kódu alebo prostredníctvom dekorátorov funkcií.

Optimálnym navrhovaným nástrojom sa zdá byť použitie novej OpenAPI verzie 3.0, ktorá je kompatibilná aj s ďalšími nástrojmi pre správu API, a teda vhodnejšia pre ďalší vývoj ako staršie verzie. Predošlé verzie podporovali vytvorenie špecifikácie priamo z kódu bez dekorátorov, pri novej verzii 3.0 je potrebné časti API špecifikovať pomocou týchto dekorátorov.

Zmeny je potrebné naraz vykonať v obidvoch moduloch serverovej strany. Keďže sa táto práca zaoberá prioritne verejnou časťou, budú zmeny vykonané paralelne s autorom prebiehajúcej práce [34], ktorý zabezpečí doplnenie potrebných náležitostí v administračnom module podľa navrhnutého vzoru v tejto práci.

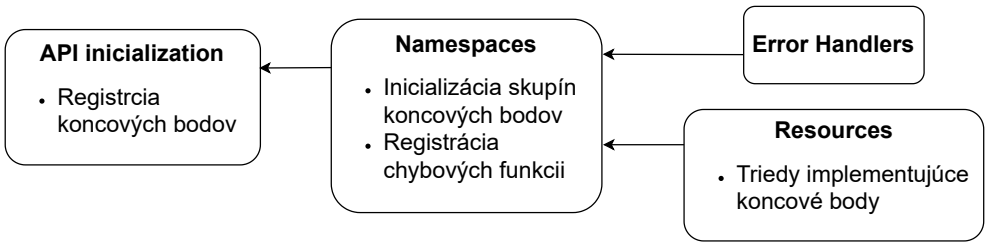
Súčasná používajúca knižnica `Flask-RESTX`, ktorá je používaná pri inicializácii API, ale nepodporuje túto verziu OpenAPI. Bude preto potrebné túto knižnicu nahradiť inou, ktorá danú verziu bude podporovať. Administračný aj verejný modul budú obsahovať zvlášť miesto, kde budú vytvorené triedy skupín, do ktorých budú koncové body pridávané podľa funkčnosti. Tieto skupiny koncových bodov budú nasledujúce:

- **inicializačné** – inicializačné koncové body ako informácie o aplikácii,
- **všeobecné** – dáta o aplikácii, členoch a funkčnom období,
- **municipalitné** – všeobecné dáta spojené s municipalitami,
- **zasadanie** – dáta zo zasadaní,
- **hlasovanie** – dáta o hlasovaniach a ich detailoch,
- **politický subjekt** – dáta k politickým stranám,
- **špecifické pre klienta** – koncové body špecifické pre potreby klienta v komponente *dropdown*,
- **analýza** – štatistické dáta, porovnanie zastupiteľov a analýza dochádzky.

Konkrétne rozdelenie koncových bodov je možné vidieť v prílohe A. Do týchto skupín budú koncové body pridané práve podľa dekorátorov, ktoré budú prístupné v novo používanej knižnici.

V predošlom riešení bola používaná trieda `ExtendedApi` pre registráciu chybových funkcií. V novom riešení bude aplikácia inicializovaná priamo z triedy `Api`. Funkcie implementujúce spracovanie chýb spoločné pre moduly budú presunuté do zvlášť súboru, odkiaľ

budú registrované do API. Každý modul bude mať navyše v časti `util` definované chybové funkcie špecifické pre daný modul.



Obr. 5.4: Schéma zobrazujúce novú inicializáciu API a jej koncových bodov. V zvlásť časti pred inicializáciou sa vytvoria menne priestory respektíve skupiny do ktorých sú pridávané koncové body. V tejto časti sú tiež registrované metódy, ktoré obsluhujú chyby za behu.

5.3 Návrh riešenia problémov z analýzy

Ako bolo spomenuté v sekcii 4.2, v systéme sa nachádzajú problémy, ktoré boli analyzované a je potrebné ich v novej architektúre vyriešiť. Navrhované zmeny sú opísané vzhľadom na vytvorené číslované referencie z kapitoly s analýzou.

V celkovom návrhu novej architektúry sa po novom bude používať volanie databázy prostredníctvom ORM. Cieľom je eliminovať používanie triedy `DatabaseHandler` a využívať prístup implementovaný v triede `ORMHandler`, čo rieši problém č. 1. Prvá trieda sa okrem súborov súvisiacich s novou architektúrou používa aj ojedinele v testoch a v administratívnej časti na niektorých miestach. Je nutné poznamenať, že trieda `ORMHandler` volá niektoré funkcie a podtriedy z `DatabaseHandler` ako napríklad `NotFoundException`. Po novom by boli využívané funkcionality presunuté iba do jednej triedy, ktorú by využívali aj zvyšné komponenty.

Problém č. 2 súvisí so staršou verziou knižnice `SQLAlchemy`. Ako už bolo spomenuté, je potrebné povýšiť verziu tejto knižnice, avšak počas analýzy bolo zistené, že novšia verzia nie je kompatibilná so súčasnou verziou akou sú implementované entity tabuliek v zložke `orm`. Problém bol opísaný v dokumentácii samotnej knižnice [61], kde verzia `SQLAlchemy 2.0` zavádza nový systém definovania tabuliek pomocou tzv. anotovaných deklaratívnych tabuliek. Tento systém získava informácie o ORM atribútoch z anotácií podľa štandardu `PEP 484`, ktoré sú uvedené priamo v definíciách tried počas behu programu. Podmienkou tohto prístupu je, že všetky ORM anotácie musia používať generický kontajner s názvom `Mapped`, aby boli správne rozpoznané. Predošlé riešenie používa starší typ príkazu `relationship`, ktorý tento generický kontajner `Mapped` nevyužíva. Navrhovaným dočasným riešením tohto problému je podľa dokumentácie pridanie označenia `__allow_unmapped__` do tried, ktoré implementujú tabuľky databáz.

V analyzovanej verzii kódu sa nachádzajú časti, ktoré používajú tabuľku `Alias`, ktorá je prázdna, a tabuľku `AliasJmeno`, v ktorej sa nachádzajú používané dáta, čo referuje na problém č. 3. Po novom budú časti kódu, ktoré používajú tabuľku `Alias` odstránené, keďže tabuľka už žiadne dáta neobsahuje, nenaruší sa funkcionalita a odstráni sa tak nevyužitý kód. Ďalší problém však nastáva pri migračných skriptoch používaných pre presun dát z jednej tabuľky do druhej. Tento skript je stále používaný a je implementovaný s použitím ORM prístupu, ktorý vyžaduje prítomnosť triedy, ktorá modeluje tabuľku `Alias`. Pre

ostatné nepoužívané tabuľky sú vytvorené triedy, ktoré ich reprezentujú. Po novom budú aj tieto triedy odstránené.

Počítanie dochádzky doteraz prebiehalo na klientskej strane (problém č. 4), pričom je žiadúce túto zodpovednosť preniesť na server a klientovi posielať finálne hodnoty. Klient bude brať výpočet dochádzky z aktualizovanej odpovede servera s atribútom obsahujúcim túto hodnotu. Výpočet bude v rámci servera presunutý do zvlášť funkcie v zložke `util`, pretože sa táto funkcia bude využívať aj v ďalších prípadoch ako dochádzka jednotlivca alebo porovnanie zastupiteľov.

Jedným z požiadavkov na rozšírenie systému, popísaný tiež v analýze problémov č. 7, je podpora viacdňových zasadaní. Pre tento účel bude rozšírená tabuľka `Zasedani`. V súčasnom stave je podporovaný na serverovej časti dátum, kedy sa zasadnutie konalo. Nasledujúci návrh zmien bol vytvorený v spolupráci s tímom Zastupko.cz. Keďže je potrebné v niektorých prípadoch uchovávať viacdňové zasadnutie, treba túto tabuľku rozšíriť o atribúty `datum_od` a `datum_do`. Aby bolo možné modelovať hlasovania počas tohto viacdňového hlasovania, je potrebné zmeniť typ atribútu `cas` z `time` na `datetime`, čím sa zabezpečí príslušnosť k časovému úseku daného zasadania. Pre modelovanie vedenia zastupiteľstva bude pridaná nová tabuľka `Vedeni`, ktorá bude obsahovať nasledujúce atribúty:

- `id` – identifikátor vedenia,
- `ck_zastupitelstvo` – cudzí kľúč odkazujúci na funkčné obdobie reprezentovaného v tabuľke `Zastupitelstvo`,
- `ck_clen` – cudzí kľuč odkazujúci na konkrétneho člena ktorý je vo vedení,
- `funkce` – konkrétna funkcia člena vo vedení,
- `datum_od` – začiatok obdobia kedy je člen vo vedení,
- `datum_do` – koniec obdobia kedy je člen vo vedení.

Ďalším problémom a zároveň požiadavkou z bodu č. 8 je, aby štatistiky a dáta boli rozdelené u členov podľa strán, ak mali členstvo vo viac ako jednej. Návrh zmien sa predovšetkým týka koncových bodov detailu zastupiteľa `/members/<int:member_id>` a analýzy dochádzky `/attendanceDashboard`. Pre túto zmenu je najprv potrebné pridať stĺpec do tabuľky `StatistikaClen`, ktorý by reprezentoval cudzí kľúč odkazujúci na identifikátor strany. Tabuľka by potom obsahovala nasledujúce atribúty:

- `id` – identifikátor štatistiky,
- `ck_zastupitelstvo` – cudzí kľuč odkazujúci na konkrétne zastupiteľstvo,
- `ck_clen` – cudzí kľuč odkazujúci na člena ku ktorému sa štatistika vzťahuje,
- `ck_moznost` – hlasovacia možnosť za ktorú je počítaná štatistika,
- `sum` – súčet hlasovaní za konkrétnu hlasovaciu možnosť,
- `ck_subjekt` – cudzí kľuč odkazujúci na politickú stranu za ktorú člen sú zaznamenané štatistiky hlasovania člena.

Ako už bolo spomínané v analýze, táto tabuľka predstavuje istú dočasnú pamäť, ktorá je pri manipulácii s dátami aktualizovaná. Je preto potrebné vykonať zmeny v administračnom module, ktorý aktualizuje túto tabuľku pri manipulácii s hlasovacími dátami. Zmena bude vykonaná v časti, ktorá komunikuje s databázou vo funkcii `update_statistics`, ktorá najprv získa dáta z aktuálnej tabuľky a zmeny aktualizuje. Z týchto krokov tiež vyplýva, že je potrebné vytvoriť migračný skript, ktorý odstráni predošlé dáta z tabuľky `StatistikaClen`, pridá stĺpec s cudzím kľúčom na tabuľku s politickými subjektmi (stranami) a naplní ju novými, už rozdelenými podľa strán.

Ďalšou navrhovanou zmenou vyplývajúcou z požiadaviek klientskej strany na server. S paralelne vznikajúcou bakalárskou prácou [22], ktorá sa zameriava na klientskú stranu informačného systému, vznikol požiadavok na zobrazenie členov danej strany podľa toho, či sú aktívni alebo neaktívni, alebo či sú členmi inej strany (problém č. 8). Vzhľadom na to bude upravená odpoveď koncového bodu `/party/<int:party_id>`. Po novom bude v zozname členov prítomný atribút `active_member`, ktorý bude obsahovať hodnoty 1 alebo 0 podľa toho, či je člen aktívny v danom funkčnom období. Ak je člen v inej politickej strane, bude prítomný atribút `political_party` obsahujúci skratku a farbu politickej strany, v ktorej bol naposledy.

Ďalšou takouto požiadavkou je pridanie informácie o pohlaví k dátam o členoch. Tento príznak bol doteraz vyžadovaný iba v prípade detailu zastupiteľa. Po novom sa tento príznak (0 - muž , 1 - žena) bude posielať v odpovediach vždy aj s menom člena pod atribútom `pohlavie`. Ďalšou menšou navrhovanou zmenou je premenovanie atribútu pri kohézii hlasovania. Pôvodne sa jednalo o atribút s názvom `ai`. Novým navrhovaným názvom bude `vote_match`. Zmena bude vykonaná v triede `VoteQuery`.

Kapitola 6

Implementácia

Hlavným cieľom implementovaných zmien bolo vytvorenie architektúry serverovej časti a jeho aplikačného rozhrania pre verejný modul systému. Ďalším cieľom bolo integrovanie nástroja pre vytváranie špecifikácie API. Táto kapitola obsahuje postup, ktorý viedol k vytvoreniu konečného riešenia na základe návrhu z kapitoly 5 a navrhovaných modifikácií pre problémy analyzované v informačnom systéme. Popísané sú hlavne netriviálne vykonané zmeny.

Keďže táto práca stavia na existujúcom riešení [68], ktoré bolo neskôr rozšírené [33], hlavná implementácia bola značne obmedzená technológiami. Hlavným používaným frameworkom je **Flask** v jazyku Python. Pre relačnú databázu je zachovaná **MariaDB** ako aj v predošlom riešení. Novo použité technológie a knižnice potrebné pri implementovaní navrhnutého riešenia sú opísané v ďalších sekciách.

6.1 Zmeny architektúry

Zmeny a zavedenie navrhovanej architektúry prebehli vo verejnej časti informačného systému. Implementácia prebehla podľa vytvoreného návrhu z podkapitoly 5.1. Všetky časti súvisiace s verejnou časťou boli presunuté do jednej zložky `public`, kde sa delia na tri časti:

- `/public/resources` – do tejto zložky boli presunuté implementácie tried koncových bodov a všetka aplikačná logika modulu,
- `/public/services` – tu boli presunuté vytvárané požiadavky na databázu (v tejto časti sa primárne získavajú dáta z databázy),
- `/public/util` – pomocné funkcie spoločné pre viac tried koncových bodov.

Názvy boli vytvorené tak, aby korešpondovali so zložkami v administračnej časti. Podľa novo presunutých tried koncových bodov bolo potrebné zmeniť ich importovanie v inicializačnom súbore serverovej časti `create_app.py`.

Vedľajšou súčasťou architektúry verejného modulu bolo vytvorenie štruktúrovanejšieho rozmiestnenia schém, ktoré implementujú tabuľky databáz. Implementácia bola vytvorená podľa návrhu na obrázku 5.2 a triedy boli rozdelené do novo vytvorených zložiek `orm/schema/smunicipality_db` a `orm/schema/central_db`. Názvy súborov boli ponechané v českom jazyku, aby bolo jednoznačné, ktorý súbor reprezentuje akú entitu. Konkrétne rozdelenie do súborov je zobrazené v prílohe B.3. Keďže predtým boli všetky schémy obsiahnuté v jednom súbore, nebolo potrebné riešiť ich vzájomné importovanie. Taktiež bolo potrebné vyriešiť doplnenie vzťahov medzi jednotlivými triedami implementujúce schémy tabuliek.

Vzťahy medzi triedami sú riešené prostredníctvom metódy `relationship` z knižnice `SQLALchemy`. Táto metóda prijíma ako argument názov druhej triedy, s ktorou je vytvorený vzťah. Keďže sa triedy tabuliek oddelili do zvlášť súborov, bolo potrebné volanie tejto metódy pridať do oboch tried v danom vzťahu. V tejto metóde bol využitý parameter `back_populates` na definovanie obojsmerného vzťahu medzi triedami. Jednoznačne sa tým určí vzťah a zabezpečí synchrónnosť.

Čo sa týka problému importovania tried v súbore iných tried, nastávala inkonzistencia s viacpočetným importom a dochádzalo k chybám v serverovej časti. Tento problém bol vyriešený vytvorením súboru `__init__.py` v obidvoch častiach. Do tohto súboru boli presunuté importy všetkých tried danej zložky a vytvorenie deklaratívneho základu pre tieto triedy. Ďalej schémy entít obidvoch zložiek používali spoločné globálne parametre. Tie boli presunuté do spoločne využívaného súboru `core.py` v zložke `orm/schema` . Rozdelenie týchto tried bolo ďalej potrebné pre implementáciu novej serverovej časti.

Opačný proces bol vykonaný v prípade tried, ktoré implementujú koncové body. Boli spojené náležitosti, ktoré spolu logicky súviseli. V nasledujúcom zozname sú triedy, ktoré boli spojené do jedného súboru:

- `Party.py` – triedy `PartiesBasicData`, `PartiesStats`, `PartiesUnity` a `PartyData`,
- `Session.py` – triedy `SessionsData`, `SessionData` a `SessionAttendance`,
- `Vote.py` – triedy `VoteData`, `VotesBasicData` a `VotesQuery`.

Triedy koncových bodov vytvorené pre potreby klientskej strany boli presunuté do zvlášť zložky `/public/resources/dropdowns`. V tejto zložke boli ponechané dva súbory, kde boli rozdelené tieto typy tried následovne:

- `FE_DropdownOptions.py` – sem boli presunuté všetky triedy, ktoré v odpovedi vracajú zoznam základných entít,
- `FE_DropdownSessionResults.py` – vracia odpoveď s možnými výsledkami hlasovania.

Jediný koncový bod ponechaný mimo štruktúru novej architektúry je `/hello`, ktorý je čisto inicializačný a nekomunikuje s databázou.

V zložke `/service`, kde boli premiestnené volania databázy, boli vytvorené triedy rozdelené jednotlivo do súborov (príloha B.2). Triedy obsahujú metódy, ktoré sú volané časťami zo zložky `/resources`. Nižšie sú opísané nové triedy a ich funkcie:

- `AttendanceService` – volania databáz pre získavanie analytických dát dochádzky členov v zasadaniach a hlasovaniach,
- `CouncilsServices` – volania databázy pre získavanie dát spojené primárne s funkčným obodbím a dátami z tabulky `Zastupitelstvo`,
- `MembersServices` – získavanie dát, ktoré súvisia primárne s členmi a ich členstvami v politických subjektoch,
- `MunicipalityServices` – základné dáta orgánov municipality,
- `SessionsServices` – získavanie dát zo zasadaní a prítomnosti členov na zasadaniach,
- `SubjectServices` – získavanie dát súvisiacich s politickými stranami a ich mandátmi,

- `VotesServices` – získavanie dát o hlasovaniach, ich výsledkoch a niektorých štatistík.

Predtým boli volania databázy pre každý koncový bod individuálne. Vďaka rozdeleniu do týchto tried je možné volať potrebné metódy z ostatných častí kódu a využiť ich tak na viac miestach.

Niektoré triedy koncových bodov obsahovali dlhší kód pre implementáciu aplikačnej logiky (triedy `MemberData` a `Dataset`). V rámci týchto tried boli vytvorené menšie funkcie, ktoré implementovali jednotlivé dielčie časti odpovede koncového bodu. Toto rozdelenie na viaceré funkcie je užitočné aj v prípade, ak časť odpovede koncového bodu je potrebná v inom koncovom bode.

Podľa návrhu bol ďalej implementovaný nový koncový bod v triede `Search`, ktorý umožňuje vyhľadávanie v troch tabuľkách `Clen`, `Hlasovani` a `PolitickySubjekt`, podľa zvolených atribútov týchto tabuliek tak, ako boli definované v návrhu. Pre implementáciu už bol použitý ORM prístup s využitím knižnice `SQLAlchemy` a z nej definovaných inštancií `or_` a `func`. Tieto dve inštancie boli použité na reprezentovanie logických operácií, ako je konjunkcia. Druhá metóda `func` bola použitá pre vytvorenie konkatenácie v prípade mena a priezviska člena, aby bolo možné vyhľadať podľa celého mena člena.

Vyhľadávanie bolo implementované tak, aby boli vrátené výsledky, ktoré vyhľadávaný reťazec obsahujú ako prefix, sufix alebo je prítomný uprostred celkového reťazca. Pri zavedení zmien bol odhalený problém vzniknutý v spojitosti s českým jazykom. V prvotnej implementácii boli rozlišované písmená obsahujúce mäkčeň, to znamená, že vyhľadávané slová ako „Katerina“ a „Kateřina“ boli programom chápané ako rozdielne. Keďže je pravdepodobné, že vyhľadávaný reťazec nebude vždy obsahovať diakritiku, bolo potrebné tento problém riešiť. Bolo preto použité porovnanie , ktoré dokáže porovnať iba jednotlivé znaky – `collate("utf8_general_ci").ilike(f"%{search_word}%")`. Vďaka tomu bolo zabezpečené presnejšie vyhľadávanie aj bez diakritiky.

Prechod na ORM

Zásadnou zmenou v rámci novej architektúry bolo zavedenie objektovo-relačného mapovania pri získavaní dát z databázy a ich manipuláciou počas spracovania. Technológia tohto prístupu už je používaná v administračnom module. Pre inicializáciu tohto prístupu je použitá existujúca trieda `ORMHandler`, implementovaná v zvlášť zložke `/orm` s použitím knižnice `SQLAlchemy`.

Táto implementácia zároveň rieši bod č. 1 z analýzy. Pre získavanie dát a spojenie s databázou je vo verejnom module použitý iba ORM prístup využívajúci spomínanú triedu. Z triedy `DatabaseHandler` boli presunuté definície výnimiek a funkcia `get_db_name` do triedy `ORMHandler`. Následne bolo možné ju odstrániť.

Rovnaká knižnica je použitá vo vytvorenej architektúre pri triedach komunikujúcich s databázou v zložke `/service` ako to bolo opísané v predošlej podkapitole. Hlavné problémy vznikali pri namapovaní niektorých častí používaných v SQL na ORM prístup. Knižnica `SQLAlchemy` poskytuje riešenie na väčšinu týchto problémov prostredníctvom inštancie `func`. Jedná sa o špeciálnu konštrukciu, ktorá poskytuje prístup k SQL funkciám. V implementácii je využívaná pre dynamické generovanie týchto funkcií. Implementácia používa nasledujúce funkcie tejto inštancie:

- `func.if_` – definovanie podmienky,
- `func.sum` – analogicky k metóde `SUM` v SQL, súčet hodnôt zadaného atribútu entity,

- `func.count` – analogicky k metóde `COUNT` v SQL, počet hodnôt zadaného atribútu entity,
- `func.date_format / func.time_format` – nastavenie formátu v prípade atribútov ktoré predstavovali typ čas alebo dátum,
- `func.coalesce` – použité v prípade keď bola vo vnútri požiadavku potrebné definovať ďalší požiadavok,
- `func.isnull` – zistenie či daný atribút obsahuje hodnotu,
- `func.min / func.max` – analogicky k metóde `MAX` a `MIN` v SQL, získanie maximálnej/minimálnej hodnoty atribútu,
- `func.avg` – analogicky k metóde `AVG` v SQL, vypočítanie priemerov hodnôt zadaného atribútu,
- `func.greatest` – analogicky k metóde `GREATEST` v SQL, výber najväčšej hodnoty zo zadaných.

Problém tiež nastáva pri type požiadavku, ktorý obsahuje v sebe externý požiadavok. ORM prístup sprístupňuje prehľadnejšie riešenie, kedy je možné definovať vnútorný požiadavok bokom v druhej premennej. Importovaná trieda z využívanej knižnice `Session`, z ktorej sú vytvárané tieto ORM požiadavky, poskytuje metódu `subquery`, vďaka čomu je možné definovať tento externý požiadavok, umiestnený vo vnútri iného.

6.2 Integrácia OpenAPI 3.0

Pôvodné aplikačné rozhranie bolo implementované s použitím knižnice `Flask-RESTful`[1], kde sa jednotlivé koncové body registrovali do API bez rozdelenia do menných priestorov. Táto knižnica nie je najaktuálnejšia a nepodporuje generovanie špecifikácie API z kódu alebo použitím dekorátorov, čo súvisí aj s problémom opísaným pod bodom č. 5. Preto bolo navrhnuté túto knižnicu nahradiť aktuálnejšou.

Pre implementáciu bola na tento účel vybraná knižnica `Flask-Smorest`[2]. Táto knižnica oproti predošlej podporuje vytváranie špecifikácie OpenAPI a interaktívnej dokumentácie. Keďže je táto knižnica naďalej vyvíjaná a aktualizovaná, predstavuje tiež možnosť, ako udržiavať aktuálny samotný informačný systém. Podpora novšej verzie OpenAPI zabezpečuje taktiež ďalšie použitie nástrojov pre vývoj API, ktoré fungujú na základe tejto špecifikácie. Implementáciu bolo potrebné realizovať v obidvoch moduloch informačného systému. Zmeny potrebné pre administračnom module boli vykonané v paralelne prebiehajúcej práci [34], kde budú aj popísané zmeny v danej veci pre tento modul.

Táto knižnica funguje na inom princípe ako predošlá, jej používanie však zapadá do návrhu podľa podkapitoly 5.2. Rozdelenie do menných priestorov je realizované prostredníctvom triedy `Blueprint`, v ktorej sú definované spoločné prefixy koncových bodov danej skupiny a ich popis. Jednotlivé inštancie tejto triedy sú definované v súbore `blueprints.py`. Menné priestory boli vytvorené podľa návrhu a sú naznačené v prílohe A. Tieto inštancie sú ďalej vložené do zoznamu a importované do inicializačnej časti API, kde sú registrované pomocou funkcie `register_blueprint`.

[1] Dostupné z: github.com/flask-restful/flask-restful
[2] Dostupné z: github.com/marshmallow-code/flask-smorest

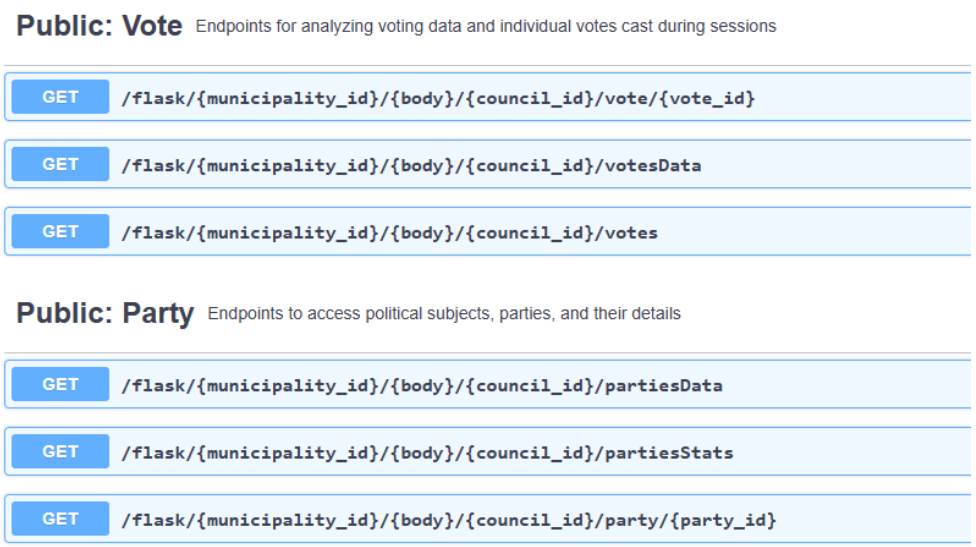
Pridanie samotných koncových bodov do týchto skupín je s použitím tejto knižnice realizované prostredníctvom dekorátorov tried koncových bodov a ich funkcií, ktoré definujú HTTP metódu. Hlavnými využívanými dekorátormi vo verejnom module boli nasledujúce:

- `@generals_blp.route` – definovanie URL koncového bodu,
- `@general_blp.arguments` – vloženie triedy reprezentujúca voliteľné argumenty,
- `@vote_blp.response` – nastavenie východzej HTTP odpovede koncového bodu.

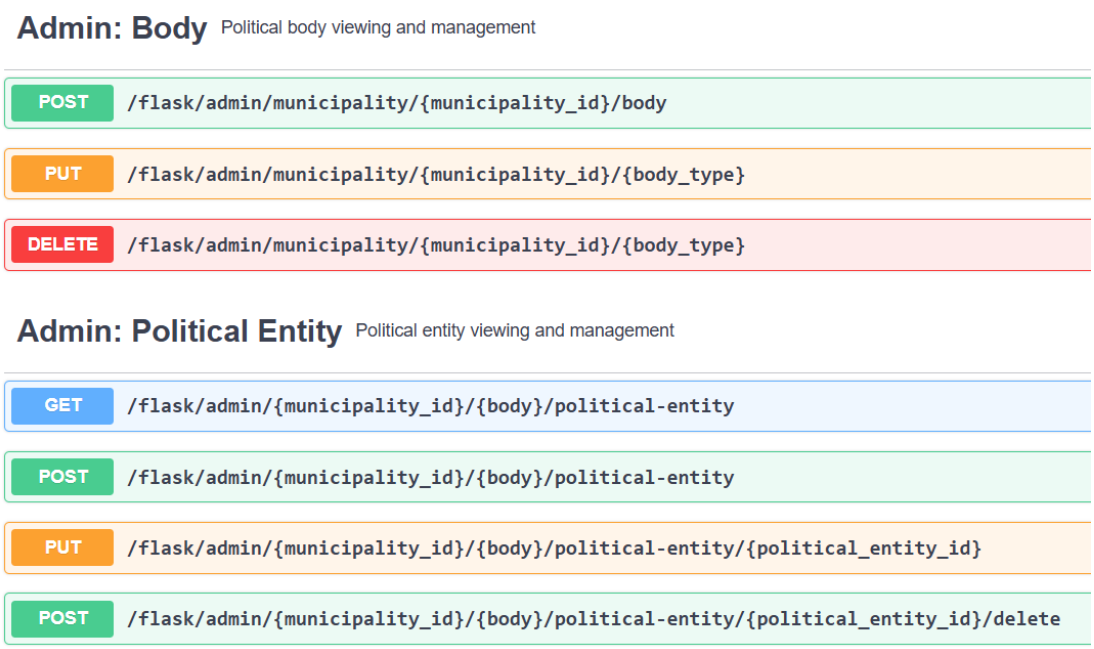
V prípade voliteľných argumentov bolo potrebné vytvoriť triedu, ktorá definuje voliteľné parametre koncového bodu. V tomto mieste je pre špecifikáciu argumentov koncového bodu API definovaný typ, popis a príznak, či sú vyžadovanými argumentami. Táto trieda je vytvorená dedením z triedy `Schema` prístupnej z knižnice `marshmallow` na ktorej funguje aj novo zvolená knižnica pre implementáciu špecifikácie API.

Knižnica `Flask-Smorest` pracuje nad tzv. metodickými pohľadmi implementovanými v samotnej knižnici `Flask` prostredníctvom triedy `MethodView`. Táto knižnica je v implementácii použitá ako dedičná trieda pre všetky triedy implementujúce koncové body API. Ďalším dôležitým aspektom bolo pridanie funkcií, ktoré obstarávajú chyby za behu. Spoločne chybové funkcie pre oba moduly boli oddelené do súboru `error_handlers.py`. Z tohto súboru sú importované a registrované spolu s mennými priestormi koncových bodov v súbore `blueprints.py`. Vďaka tomu budú zobrazované správne chybové kódy serverovej strany.

Po implementovaní všetkých náležitostí pre správne fungovanie API bola automaticky z tejto špecifikácie vygenerovaná interaktívna dokumentácia aplikačného rozhrania pre obidva moduly. Cesta, na ktorej je možné k nej prestupovať, bola určená takto `/flask/doc`. V užívateľskom rozhraní bolo potrebné rozlíšiť menné priestory, ktoré patria do verejného a ktoré do administračného modulu. Keďže špecifikácia neponúka ich vyššie rozdelenie, boli použité prefixy „Public“ a „Admin“, pre rozdelenie menných priestorov týchto modulov (obrázok 6.1 a 6.2).



Obr. 6.1: Podla prefixu v názve menného priestoru je vidieť, že sa jedná o verejný modul, pri čom každý tento priestor obsahuje tri koncové body.



Obr. 6.2: Podla prefixu v názve menného priestoru je vidieť, že sa jedná o administračný modul, pri čom prvý menný priestor (Body) obsahuje tri koncové body a druhý menný priestor (Political Entity) obsahuje štyri. Farebne sú v UI odlíšené typy HTTP požiadavkov.

6.3 Riešenia požiadavkov a problémov

Táto sekcia popisuje implementáciu zmien navrhnutých v podkapitole 5.3 k analyzovaným nedostatkom systému. Odkazovanie na definované problémy bude použité tak, ako bolo zavedené v podkapitole 4.2.

Ako prvé bolo implementované riešenie problému č. 2, kde sa jednalo o prítomnosť staršej verzie knižnice `SQLALchemy` na ktorej stavali aj upravované časti tohto systému. Knižnicu najprv nebolo možné povýšiť na vyššiu verziu z dôvodu nekompatibility medzi novo zavedenými štruktúrami tejto knižnice a predošlou verziou kódu informačného systému. Ako bolo navrhnuté, tento problém kompaktnosti bol vyriešený pridaním `__allow_unmapped__`, ktorého hodnotu bolo potrebné nastaviť na `True`. Vďaka tomu bolo možné do prostredia integrovať najnovšie verzie tejto knižnice.

Pri vytváraní novej architektúry a zmenách tried koncových bodov boli odstránené časti kódu, ktoré súviseli s prázdnou tabuľkou `Alias` a ponechané iba tie, ktoré pracovali s tabuľkou `AliasJmeno`. Odstránenie týchto častí kódu prebehlo vo všetkých triedach, ktoré vo svojej odpovedi vracali zoznam členov alebo detail jedného člena. S tým aj súviselo odstránenie ďalších nepoužívaných tabuliek z databázy, ako popisuje bod č. 3. Pre tento účel bol vytvorený migračný skript `remove_unused_tables.py` v zložke `/migrations`, s použitím triedy `ORMHandler`.

Ďalším analyzovaným problémom bol bod č. 4, kde podľa návrhu bolo potrebné odstrániť výpočet niektorých hodnôt z klientskej časti a presunúť túto zodpovednosť na server. Najskôr boli tieto časti identifikované a výpočet na klientskej strane odstránený s tým, že bol nahradený patričnou hodnotou v odpovedi zo strany servera. Hodnota bola uložená pod novým kľúčom `attendance` alebo `attendance_stat`. V prípade koncového bodu, ktorý

implementuje logiku pre porovnanie dát dvoch zastupiteľov, bol tiež presunutý výpočet percentuálnej zhody z klienta do triedy koncového bodu, prístupná pod kľúčom `match_stat`.

Samotný výpočet je univerzálny pre obidva prípady, preto bola pridaná funkcia do časti `util` podľa implementovanej architektúry s názvom `count_attendance`. Táto funkcia prijíma tri vstupné parametre, z toho dva sú povinné. Prvý predstavuje počet prítomných (počet zhôd v prípade porovnania) a druhý predstavuje celkový počet hlasovaní. Tretí parameter udáva, na koľko desatinných miest má byť výsledok zaokrúhlený. Východzie nastavenie tohto parametru je zaokrúhľovanie na dve desatinné miesta. Toto nastavenie je implementované z dôvodu vzniku problému opísaného pod bodom č. 9. Výpočet ďalej prebieha ako podiel prvých dvoch parametrov upravený do percentuálnej podoby. Klient tak dostáva od servera hodnotu priamo v odpovedi a iba ju dáva na výstup užívateľovi.

Počas vytváraných úprav na klientskej strane s paralelne vznikajúcou prácou [22], bol pridaný požiadavok na serverovú časť, kde bolo potrebné v detaile politického subjektu a zozname jej členov identifikovať tých, ktorí už v danej strane nie sú, ale stále sú aktívni v inej. Podobné dáta však už poskytovala existujúca trieda koncového bodu detailu člena a to `MemberData`. Ako bolo spomenuté v predchádzajúcej podkapitole implementácie, niektoré triedy a ich logika boli rozdelené do viac menších funkcií, ktoré implementujú jednotlivé časti koncového bodu. Pre získanie dát o histórii členstiev v stranách bola využitá trieda `MemberData` a jej dielčia metóda `_populate_political_parties`. Vďaka tomu boli získané politické strany, v ktorých bol zastupiteľ členom, a následne upravená odpoveď pre detail subjektu. Názvy kľúčov v odpovedi boli upravené podľa navrhovaného riešenia.

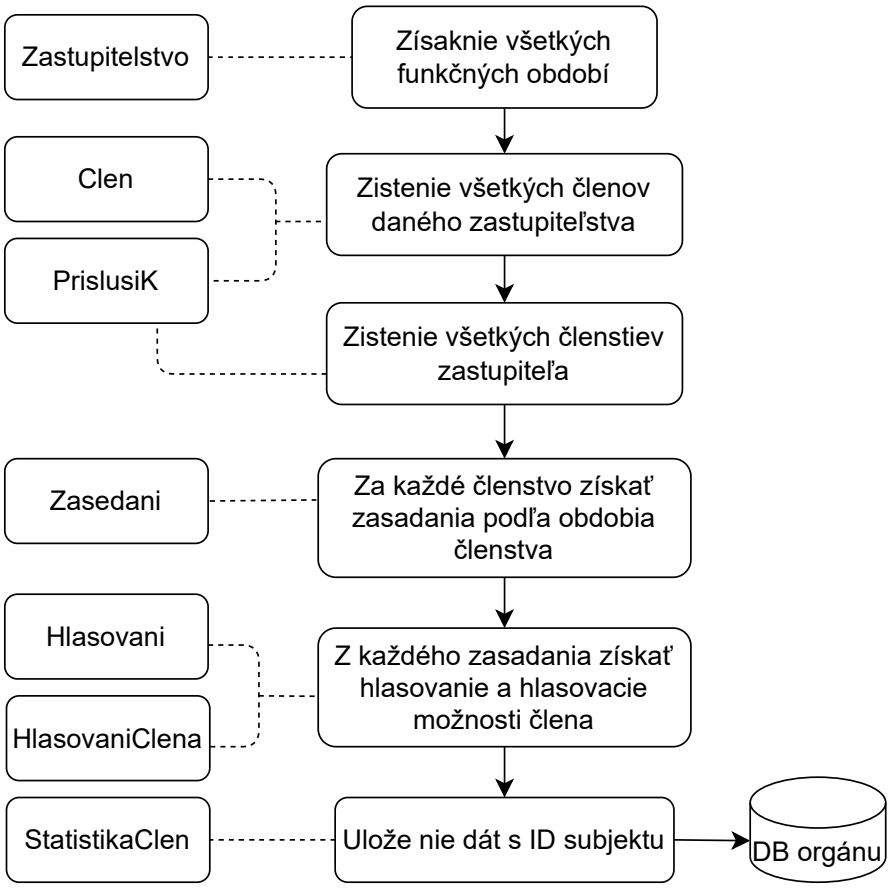
Rozdelenie dát podľa politických strán

Väčšou zmenou, ktorou bolo potrebné sa zaoberať, je riešenie analyzovaného problému č. 8, ktorý hovorí o tom, že dáta zastupiteľov nie sú rozdelené podľa strán, ak mal zastupiteľ členstvo vo viacerých. Ako bolo spomenuté v návrhu, problém sa týka predovšetkým tried `AttendanceDashboard` a `MemberData`.

Najskôr bolo potrebné vytvoriť migračný skript `stats_per_subject.py`, fungujúci tak, že najprv prebehne odstránenie všetkých doterajších dát v tabuľke `StatistikaClen`. Následne je pridaný stĺpec pre cudzie kľúče odkazujúce na identifikátor politického subjektu. Ďalej je potrebné z ostatných tabuliek získať patričné dáta a vložiť ich do novej verzie tejto tabuľky. Najprv sú zistené funkčné obdobia, za ktoré sú dáta v orgáne zaznamenávané. Následne je potrebné zistiť identifikátory všetkých členov v tomto období a ich jednotlivé členstvá. Následne podľa členstiev je zistený časový úsek, za ktorý mal člen v danom období mandát. Podľa toho je možné získať identifikátory zasadaní, ktorých sa zastupiteľ počas členstva mal zúčastniť. Ako posledný krok je získanie všetkých hlasovaní počas zasadaní a aké hlasovacie možnosti boli zvolené daným zastupiteľom. Po poslednom kroku sú ukladané dáta súčtov za jednotlivé hlasovacie možnosti, ktoré zastupiteľ vykonal počas členstva v jednej strane. Potrebné štatistiky aplikačnej logiky sú potom ďalej získavané z tejto novo upravenej tabuľky. Tento postup je znázornený spolu s využitými tabuľkami na obrázku 6.3.

Ako prvé bola upravovaná logika triedy koncového bodu pre detail zastupiteľa. V tomto prípade bolo potrebné najprv zistiť presné obdobie, za ktoré bol v strane. Na tento účel bola vytvorená pomocná funkcia `populate_member_parties` v zložke `util`, pretože ten istý výpočet je potrebný aj v prípade vytvárania celkovej dátovej sady orgánu v triede `Dataset`. Účelom tejto pomocnej funkcie je vyplniť dáta v štruktúre obsahujúcej dáta strany a najmä doplnenie hodnôt začiatku a konca členstva podľa prvého a posledného zasadania

mandátu. Ďalej sa podľa tohto obdobia získajú dáta zasadaní, ktoré sú potrebné v odpovedi tohto koncového bodu. Zasadnutia sú rozdelené do troch zoznamov podľa toho, či na nich bol zastupiteľ prítomný, čiastočne prítomný alebo neprítomný. Vďaka upravenej tabuľke `StatistikaClen` bol vytvorený požiadavok, ktorý z databázy získa všetky súčty hlasovacích možností, ktoré zastupiteľ vykonal počas validných hlasovaní.



Obr. 6.3: Postup aktualizovania dát v tabuľke `StatistikaClen` podla toho za akú stranu boli zastupiteľom vykonané. V ľavej časti sú zobrazené databázové tabuľky, ktoré boli v danom kroku využite pomenované analogicky k schéme 4.5.

Bolo potrebné počas implementácie ošetriť prípad, kedy sa člen orgánu vrátil do predošlej politickej strany po tom, čo z nej odišiel. Tento problém je v kóde riešený tak, že je zaznamenávané, za akú stranu sa získavajú dáta. Ak už z danej strany boli načítané dáta, ale za rozdielne obdobie, tieto dáta sa pripoja do vznikajúcej štruktúry. Všetky tieto dáta sú v odpovedi uložené v zvlášť zozname s názvom kľúča `votes_per_party`. Štruktúra tejto novej časti odpovede je zobrazená v listingu 1. České názvy kľúčov boli ponechané kvôli zavedenému mapovaniu hodnôt na klientskej strane. Z rovnakého dôvodu boli použité aj indexy ako kľúče. Tieto časti predstavujú zoznam štruktúr zasadaní, kde indexy znamenajú následovné:

- 0 – zoznam zasadaní, na ktorých bol zastupiteľ prítomný,

- 1 – zoznam zasadaní, na ktorých bol zastupiteľ čiastočne prítomný,
- 2 – zoznam zasadaní, na ktorých bol zastupiteľ neprítomný.

Taktiež, ak niektorý zo súčtov obsahuje nulovú hodnotu alebo prázdny zoznam, nie je potom prítomný ani v posielanej odpovedi na klienta.

```
"votes_per_party": [
    {
        "party_id": 123,
        "votes_present": 700,
        "votes_total": 900,
        "attendance_stat": 77.77,
        "vote_options": {
            "ano": 650,
            "ne": 20,
            "zdržel se": 10,
            "nehlasoval": 10,
            "nepřít.": 7,
            "tajná": 3
        },
        "attendance_sum": {
            "přítomnost": 6,
            "částečná přítomnost": 2,
            "nepřítomnost": 1,
        },
        "0": [ {
            "id": 123,
            "meeting_number": "Z7/77"
            },
        ],
        "1": [],
        "2": []
    }, {}]
```

Listing 1: Ukážka novo implementovanej časti odpovede z API vo formáte JSON. Táto časť odpovede pre detail klienta predstavuje zoznam politických strán v ktorých zastupiteľ za funkčné obdobie pôsobil.

Ďalšou časťou, v ktorej bolo potrebné implementovať rozdelenie dát po stranách, je trieda `AttendanceDashboard`. V tejto triede je implementovaná logika pre stránku, ktorá zobrazuje štatistiku dát pre všetkých členov orgánu v danom funkčnom období. Znova je použitá modifikovaná tabuľka pre získavanie dát. Najprv bola upravená požiadavka na databázu tak, aby zahŕňala identifikátor strany, ku ktorej daná štatistika patrí. K tomu bolo podľa špecifikácií z klientskej strany tiež potrebné pridať aj informácie o tom, v akej je člen strane a či je aktívnym členom orgánu. V požiadavku pre implementáciu tohto prípadu bola použitá funkcia `func.if_` a externý požiadavok, ktorý sa dopytoval na členstvá. Ďalšou informáciou, ktorú bolo treba pridať k samotnej strane, bolo, či je strana aktuálne aktívna. To bolo zistené podľa toho, či strana obsahuje aktívnych členov. Ak mal tento príznak u všetkých členov hodnotu 0, bola vyhodnotená táto strana za neaktívnu. Oproti predošlej verzii odpovede pribudli do zoznamov členov strán aj tí, čo aktuálne v strane nie sú. Nové atribúty sú následovné:

- `is_active` – príznak u politického subjektu, ktorý určuje či je aktívny vo funkčnom období,
- `in_party` – príznak u zastupiteľa, ktorý určuje či je aktívny člen strany.

Ako bolo spomenuté v návrhu, dáta modifikovanej tabuľky sú aktualizované prostredníctvom administračného modulu pri manipulácii s hlasovacími dátami. Aby bolo možné aktualizovať túto modifikovanú tabuľku, bolo potrebné upraviť funkciu `update_statistics` v časti `/admin/service`. V tejto funkcii je používaný SQL prístup. Riešenie bolo vytvorené pridaním `JOIN` príkazu pre namapovanie hodnôt s tabuľkou `prislusi_k`. Do ukladaných dát bol následne pridaný aj identifikátor strany a uložený do databázy.

Kapitola 7

Testovanie

Oproti predošlej verzii systému bolo implementovaných niekoľko zmien, ktoré upravujú aplikačnú logiku systému. Táto kapitola sa zaoberá overením správnosti a funkčnosti vytvorených riešení. Pre tento účel slúžia už existujúce testy serverovej strany aplikačného rozhrania s využitím testovacieho frameworku `Pytest`, ktoré overujú oba moduly serverovej časti API. Najskôr však bola testovacia sada rozšírená, aby bolo možné overiť modifikácie a pridané funkcionality.

7.1 Modifikácia testov

Rozšírenie testovacej sady prebiehalo v zložke `api_test`, najmä v súbore `test_read.py` a zložke `/data/json`. Ako prvé bola pridaná funkcionalita, ktorá overuje správnosť novo vzniknutého koncového bodu pre vyhľadávanie v orgáne zastupiteľstva za konkrétne funkčné obdobie. Koncový bod hľadá daný reťazec v troch entitách, ako tomu bolo popísané v návrhu. Doplnené preto boli nasledujúce testovacie funkcie:

- `test_search_member` – funkcia testovanie správneho vyhľadávania mena zastupiteľa musela kontrolovať prípady kedy nebola použitá diakritika, napríklad v prípade hľadaného reťazca „Břetislav“ a „Bretislav“ by mal byť vrátený rovnaký výsledok. Ďalší prípad bolo potrebné skontrolovať vyhľadávanie celého mena teda s medzerou v zadanom reťazci;
- `test_search_voting` – funkcia pre testovanie vyhľadávania hlasovania podla jeho predmetu, bolo potrebné overiť že sa reťazec nachádza vo všetkých hlasovaniach, ktoré boli v odpovedi koncového bodu;
- `test_search_subject` – funkcia pre overenie správnosti vyhľadávania politických strán, kde bolo podstatné overiť aby prebehlo nie len podľa skratky ale aj celého názvu strany.

Vďaka týmto testovacím funkciám bolo možné overiť správnosť vytvoreného koncového bodu a neskôr zistiť celkovú korektnosť API.

Ako bolo spomenuté v analýze, testy pri kontrole požiadavkov `POST`, `PUT` a `DELETE`, vytvárajú nové tabuľky v predpripravenej databáze podľa zadaného SQL skriptu. Zmena, ktorá bola vykonaná v súvislosti s tabuľkou `StatistikaClen`, do ktorej bol pridaný nový atribút, je potrebné vykonať overenie správnosti novo vytvorených dát tejto tabuľky. Existujúci test overoval aktualizovanie dát v tejto tabuľke, ak bolo pridané hlasovanie. Aby bol test

funkčný aj po tejto modifikácii, bolo potrebné upraviť skript `zastupitelstvo_brno.sql` podľa ktorého sa vytvárali tabuľky v dočasnej testovacej databáze (po prebehnutí testu sú dáta tejto databázy odstránené).

Testy verejného modulu prebiehajú prevažne kontrolou odpovede daného koncového bodu tak, že sa porovná s referenčným súborom typu JSON reprezentujúcim očakávanú odpoveď. Keďže sa menili názvy atribútov niektorých verzií, bolo potrebné manuálne zmeniť tieto referenčné súbory. Ak sa odpoveď nezmenila, bol súbor ponechaný v pôvodnom stave.

7.2 Overenie správnosti zavedenej architektúry

Nemodifikované testy slúžili na začiatku vývoja k overeniu správnosti koncových bodov API po zmene architektúry vo verejnom module. Počiatočné testy tiež overovali správnosť dát počas prechodu z SQL na ORM prístup, kde bolo kľúčové overiť, že namapovanie dát prebehlo úspešne v porovnaní s očakávaným výstupom. Po implementácii štruktúry boli ďalej pridávané nové funkcionality a zmeny v odpovediach koncových bodov API. V tomto bode vývoja už boli použité modifikované testy, ktoré overujú tieto zmeny. Behy testovacích skriptov uspeli, čo preukázalo správnosť výstupov koncových bodov API.

Bolo potrebné tiež overiť novo vložené dáta do tabuľky `StatistikaClen`. Priložené testy však kontrolujú len aktualizáciu týchto dát v rámci manipulácie s hlasovaniami. Neoverujú však celkovú správnosť týchto dát. V prípade detailu zastupiteľa bola pre tento účel vytvorená funkcia `_check_sums`. Funkcia slúži pre overenie, že súčet jednotlivých hlasovaní rozdelený po stranách korešponduje s celkovým počtom hlasovaní, ktoré zastupiteľ vykonal. Podobným spôsobom sú overené aj súčty dielčich hlasovacích možností a počty zasadaní. Ak súčty nesúhlasia, je použitá funkcia `current_app.logger.warning` pre vytvorenie upozornenia na tento nesúlad . Vzhľadom na to, že sa jedná o kľúčovú štatistiku, prebehlo ďalej ručné testovanie členmi tímu Zastupko.cz. To spočívalo v overení rovnosti jednotlivých súčtov viditeľných v klientskej časti aplikácie so sumarizovanými hodnotami vypočítanými nad tabuľkami obsahujúcimi pôvodné údaje, tzn. bez využitia pomocných tabuliek, ktoré ukladajú predbežné súčty.

7.3 Overenie funkčnosti s klientskou stranou

Overovanie správnosti odpovedí API bolo tiež testované spoločne s klientskou stranou vzhľadom na vzniknuté požiadavky. Testovanie prebehlo v spolupráci s autorom bakalárskej práce [22]. Overenie správnosti komponentov užívateľského rozhrania spolu s odpoveďami servera bolo vykonávané prevažne manuálne.

Toto testovanie odhalilo nedostatok triedy koncového bodu `VotesQuery`, ktorý spočíval v zlom filtrovaní hlasovaní podľa zadaných voliteľných parametrov. Konkrétne sa jednalo o vyhľadávanie v hlasovaniach podľa predmetu a filtráciu podľa toho, či bolo hlasovanie procedurálne, tajné alebo nevalidné. Pôvodné testy API tento problém neodhalili, pretože neboli overené niektoré hraničné situácie. Spôsob implementácie bol ponechaný z predošlej verzie systému. Pre opravu bol nahradený novým spôsobom vyhľadávania, podobným aký bol zavedený aj v prípade `Search`, čím sa zabezpečila aj konzistencia vo vyhľadávaní, ktoré tieto triedy poskytujú. Pre ďalšiu opravu filtrácie bol vytvorený voliteľný parameter `zobrazit` do ktorého boli pridané argumenty podľa ktorých sa malo filtrovať. Ak prítomný nebol, počítalo sa s jeho opačnou verziou.

Ostatné náležitosti a úpravy vyžadované z klientskej strany boli manuálnym testovaním vyhodnotené ako korektné. Istým znakom korektnosti nových štruktúr odpovedí je aj to, že novo vzniknutá verzia klientskej časti tieto dáta používa a zobrazuje používateľovi. Nie je však možné túto správnosť overiť automatickým spôsobom a zabezpečiť tak neustále overenie testovacích prípadov. Pre tento účel by bolo vhodné ako príklad použitie nástroja Cypress[1] pre lepšie testovanie medzi klientskou a serverovou časťou.

[1] www.cypress.io

Kapitola 8

Záver

Cieľom práce bolo vytvoriť novú architektúru serverovej časti webového informačného systému hlasovania zastupiteľstiev a integrovať špecifikáciu aplikačného rozhrania do tohto systému pomocou vybraného nástroja. Najprv v rámci teoretickej časti bola spracovaná problematika fungovania orgánov samospráv v Českej republike a dát z ich činnosti, najmä z hlasovania orgánov samospráv. Ďalej boli analyzované rôzne prístupy k vytvoreniu architektúry serverovej časti a API. Ako ďalšie bol zanalyzovaný existujúci systém pre zobrazovanie a analýzu hlasovacích dát zastupiteľstiev a to projekt Zastupko.cz. Analyzovaná bola najmä jeho serverová časť. Prieskumom tohto projektu boli identifikované nedostatky serverovej časti verejného modulu. Následne bol vytvorený návrh architektúry, aplikačnej logiky podľa analyzovaných problémov a zmien pre integrovanie špecifikácie aplikačného rozhrania. Navrhnuté zmeny architektúry a špecifikácie boli integrované pomocou predstavených technológií.

Výsledné riešenie je v súčasnej dobe dostupné a nasadené na vývojárskej verzii tohto projektu, ktorá je verejne dostupná. Zavedenie architektúry do verejného modulu prinieslo prehľadnejšiu štruktúru a lepšiu modularitu tejto časti, vďaka čomu sa zabezpečila optimálnejšia rozširovateľnosť systému. V novej verzii boli tiež vytvorené rozšírenia, ktoré viedli k väčšej transparentnosti v spolupráci s klientskou časťou, ktoré priniesli prehľadnejšie zobrazenie dát. Integrácia špecifikácie API vniesla do systému lepšiu manuálnu testovateľnosť koncových bodov a je tiež jednoduchšie zorientovať sa v serverovej časti pre nových vývojárov tohto systému. Zmeny v tejto práci tiež riešia analyzované problémy z predošlej verzie, vďaka ktorým je zabezpečená lepšia funkčnosť celkového systému.

Pre budúce rozšírenie je doporučené zamerať sa na optimalizáciu počtu požiadaviek zo strany klienta na server. V niektorých častiach sa posielajú informácie, ktoré na klientovi nie sú potrebné.

Výsledky tejto práce zabezpečili ďalšiu možnú rozšíriteľnosť častí systému a malo by byť jednoduchšie na ne v ďalšom vývoji naviazať. Vďaka výstupu tejto práce v podobe interaktívnej dokumentácie a zavedenej architektúre bude jednoduchšie odhaliť chyby vzniknuté ďalším vývojom, a dosiahnuť tak ešte funkčnejší a spoľahlivejší systém spolu s presnejšími dátami o politických subjektoch pre občanov Českej republiky.

Literatúra

[1] Abell Gamazo, A.; Ayala Martínez, C. P.; Farré Tost, C.; Gómez Seoane, C.; Oriol Hilari, M. et al. *A data-driven approach to improve the process of data-intensive API creation and evolution*. 2017, s. 1–8. ISSN 1613-0073.

[2] Adzic, G. a Chatley, R. Serverless computing: economic and architectural impact. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2017, s. 884–889. ESEC/FSE 2017. ISBN 9781450351058. Dostupné z: https://doi.org/10.1145/3106237.3117767.

[3] Alghamdi, A.; Owda, M. a Crockett, K. Natural Language Interface to Relational Database (NLI-RDB) Through Object Relational Mapping (ORM). In: Angelov, P.; Gegov, A.; Jayne, C. a Shen, Q., ed. *Advances in Computational Intelligence Systems*. Cham: Springer International Publishing, 2017, s. 449–464. ISBN 978-3-319-46562-3.

[4] Barnes, J. M. Object-relational mapping as a persistence mechanism for object-oriented applications. *Mathematics, Statistics, and Computer Science Honors Projects*. 1. vyd., 2007, č. 6.

[5] Bauer, C. *Hibernate in action*. 1. vyd. Manning Publications Co, 2005. ISBN 1932394-15-X.

[6] Biehl, M. *RESTful Api Design*. 1. vyd. API-University Press, 2016. ISBN 978-1514735169.

[7] Blinowski, G.; Ojdowska, A. a Przybyłek, A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access*, 2022, zv. 10, s. 20357–20374. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9717259.

[8] Brito, G. a Valente, M. T. REST vs GraphQL: A Controlled Experiment. In: *2020 IEEE International Conference on Software Architecture (ICSA)*. 2020, s. 81–91.

[9] Caletka, S. Krajské volby v České republice: Regional elections in the Czech Republic. *Slovak Journal of Public Policy and Public Administration*. 1. vyd., 2014, zv. 1, č. 1.

[10] Cenková, D. *Vztahy obcí a krajů [online]*. 2016 [cit. 2024-11-04]. Dostupné z: https://is.muni.cz/th/wss8m/. SUPERVISOR : Petr Kolman.

[11] Česko. Zákon č. 1 zo dňa 1. januara 1993 o územných samosprávach. In: *Sbírka zákonů České republiky*. 1993, částka 1, s. 3–16. ISSN 1211-1244.

[12] Česko. Zákona č. 347/1997 Sb., o vytvoření vyšších územních samosprávných celků a o změně ústavního zákona České národní rady č. 1/1993 Sb. In: *Sbírka zákonů České republiky*. 1997, částka 114, s. 7018. ISSN 1211-1244.

[13] Česko. Zákon č. 106/1999 Sb., o svobodném přístupu k informacím. In:. 1999, sv. 39, s. 2578–2582. ISSN 1211-1244.

[14] Česko. Zákon č. 128 ze dne 12. dubna 2000 o obcích (obecní zřízení). In: *Sbírka zákonů České republiky*. 2000, částka 38, s. 1737–1764. ISSN 1211-1244.

[15] Česko. Zákon č. 129 ze dne 12. dubna 2000 o obcích (obecní zřízení). In: *Sbírka zákonů České republiky*. 2000, částka 38, s. 1737–1764. ISSN 1211-1244.

[16] Česko. Zákon č. 261/2021 Sb., kterým se mění některé zákony v souvislosti s další elektronizací postupů orgánů veřejné moci. In: *Sbírka zákonů České republiky*. 2021, sv. 113. ISSN 2336-517X.

[17] Chaudhary, S. Scientific study of serverless architecture, implementation and pros and cons. *Mathematical Statistician and Engineering Applications*, 2022, zv. 71, č. 1, s. 21–25. ISSN 2326-9865. Dostupné z: https://philstat.org/index.php/MSEA.

[18] Digitel. *Priame prenosy zo zasadnutí zastupitelstiev*. 2005. Dostupné z: https://www.zastupitelstvo.sk/.

[19] Docs, M. W. Promise. *JavaScript*. 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. Path: References; JavaScript; Reference; Standard built-in objects; Promise.

[20] Dubašáková, T. *Komparatívny pohľad na Asociáciu krajov Českej republiky a Združenie samosprávnych krajov SK 8*. Praha, CZ, 2020. Bakalárska práca. Vysoká škola regionálního rozvoje a Bankovní institut – AMBIS.

[21] Ehsan, A.; Abuhaliqa, M. A. M.; Catal, C. a Mishra, D. RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences*. MDPI, 2022, zv. 12, č. 9, s. 4369.

[22] Etzler, L. *Uživatelská přívětivost systému hlasování zastupitelstev*. Brno, CZ, 2025. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/161354.

[23] Fišmanová, M. *Územní samospráva-samostatná působnost krajú*. Pardubice, CZ, 2010. Bakalárska práca. Univerzita Pardubice.

[24] Fric, B. K. *Diplomová práce Územní samospráva–faktory výsledků krajských voleb v roce 2016*. Praha, CZ, 2017. Dizertačná práca. Česká zemědělská univerzita v Praze.

[25] González Mora, C.; Barros, C.; Garrigós, I.; Zubcoff, J.; Lloret, E. et al. Improving open data web API documentation through interactivity and natural language generation. *Computer Standards & Interfaces*. 4. vyd., 2023, zv. 83, č. 83, s. 103657. ISSN 0920-5489. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0920548922000344.

[26] Halačková, R. *Jednání zastupitelstva obce se zaměřením na jednací řád.* Brno, 2017. Diplomová práce. Vedúci práce Smutná, V. Dostupné z: https://is.muni.cz/th/v45ks/.

[27] Haq, S. ul. Introduction to monolithic architecture and microservices architecture. *Https://medium. com/koderlabs/introduction-to-monolithicarchitecture-and-microservices-architecture-b211a5955c63*, 2018.

[28] Hásek, R. *Metodické doporučení k činnosti územních samosprávných celků: č. 1.2 Jednací řády zastupitelstev obcí.* Ministerstvo vnitra České republiky, odbor veřejné správy, dozoru a kontroly, 2022. Dostupné z: https://mv.gov.cz/odk2/clanek/metodicke-materialy-k-zakonnym-zmocnenim.aspx.

[29] Janošík, A. *Nástroj pro zpracování dat z hlasování obecních zastupitelstev.* 2024. Bakalárska práca. VUT BRNO.

[30] Kadeřábková, J. a Peková, J. *Územní samospráva-udržitelný rozvoj a finance.* Wolters Kluwer Česká republika, 2012. ISBN 978-80-7357-263-3.

[31] Komoda, N. Service Oriented Architecture (SOA) in Industrial Systems. In: *2006 4th IEEE International Conference on Industrial Informatics.* 2006, s. 1–5.

[32] Kováčová, L. *Otevřená data - Základy otevřených dat.* 2024. Dostupné z: https://opendata.gov.cz/start.

[33] Krejčí, O. *Správa municipalit v systému hlasování městských zastupitelstev.* 2024. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií.

[34] Krejčíř, B. Štepán. *Administrační systém hlasování zastupitelstev.* Brno, CZ, 2025. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/161358.

[35] Lagu. *Roadmap to Backend Programming Master: Architectural Patterns.* 2024. Dostupné z: https://medium.com/@hanxuyang0826/roadmap-to-backend-programming-master-architectural-patterns-c763c9194414.

[36] Lawi, A.; Panggabean, B. L. E. a Yoshida, T. Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System. *Computers*, 2021, zv. 10, č. 11. ISSN 2073-431X. Dostupné z: https://www.mdpi.com/2073-431X/10/11/138.

[37] Leonard Richardson, M. A. *RESTful Web APIs.* 1. vyd. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2013. ISBN 978-1-449-35806-8.

[38] Linnik, I. *Understanding the Science Behind a High-Performing Web Application Architecture.* 2024. Dostupné z: https://softteco.com/blog/web-application-archtecture-explained.

[39] Liu, Y.; Hu, E. a Chen, X. Architecture of Information System Combining SOA and BPM. In: *2008 International Conference on Information Management, Innovation Management and Industrial Engineering.* 2008, sv. 1, s. 42–45.

[40] Magistrát hlavného mesta SR, Bratislava. *Hlasovania členov a členiek Mestského zastupiteľstva od roku 2022*. 18. júla 2023. Dostupné z: https://data.bratislava.sk/datasets/MagBa::hlasovania-%C4%8Dlenov-a-%C4%8Dleniek-mestsk%C3%A9ho-zastupite%C4%BEstva-od-roku-2022/about.

[41] Marijn Janssen, Y. C. a Zuiderwijk, A. Benefits, Adoption Barriers and Myths of Open Data and Open Government. *Information Systems Management*. Taylor & Francis, 2012, zv. 29, č. 4, s. 258–268. Dostupné z: https://doi.org/10.1080/10580530.2012.716740.

[42] Mervartová, D. *Územní aspekt místní samosprávy v České republice*. Olomouc, CZ, 2023. Bakalárska práca. Univerzita Palackého. Katedra správního a finančního práva.

[43] Ministerstvo zahraničných vecí a európskych záležitostí Slovenskej republiky. *Vyšehradská skupina*. Dostupné z: https://www.mzv.sk/diplomacia/regionalna-spolupraca/slovensko-a-v4/vysehradska-skupina. Path: Domov; Diplomacia; Regionálna spolupráca; Slovensko a V4.

[44] Mishra, S. *Software Design Principles and Architectures-a case study*. 2024. Bakalárska práca. Haaga-Helia University of Applied Sciences.

[45] Míšek, J. *Nové povinnosti pro obce, kraje a orgány státní správy v oblasti otevřených dat*. 3. augusta 2021. Dostupné z: https://data.gov.cz/%C4%8Dl%C3%A1nky/nov%C3%A9-povinnosti-pro-obce-kraje-a-org%C3%A1ny-st%C3%A1tn%C3%AD-spr%C3%A1vy-v-oblasti-otev%C5%99en%C3%BDch-dat.

[46] Nevěčná, I. *Základy veřejné správy*. Olomouc, CZ, 2013. Dizertačná práca. Univerzita Palackého v Olomouci. ISBN 978-80-244-3799-6.

[47] Nguyen, T. D.; Nguyen, A. T.; Phan, H. D. a Nguyen, T. N. Exploring API Embedding for API Usages and Applications. In: API Usages, E. A. E. for a Applications, ed. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, s. 438–449. ISSN 1558-1225.

[48] O'Neil, E. J. Object/relational mapping 2008: hibernate and the entity data model (edm). In: Computing Machinery, A. for, ed. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: [b.n.], 2008, s. 1351–1356. SIGMOD '08. ISBN 9781605581026. Dostupné z: https://doi.org/10.1145/1376616.1376773.

[49] Papučiarová, V. *Právní úprava pravomocí v České republice a na Slovensku*. 2010. Bakalárska práca. Univerzita Tomáše Bati ve Zlíne. Dostupné z: https://digilib.k.utb.cz/handle/10563/12098.

[50] Ponelat, J. a Rosenstock, L. *Designing APIs with Swagger and OpenAPI*. Simon and Schuster, 2022. ISBN 9781617296284.

[51] Postman development team. *Intro to API Platforms*. 2024. Dostupné z: https://www.postman.com/api-platform/api-documentation/.

[52] Rajan, R. A. P. Serverless Architecture - A Revolution in Cloud Computing. In: *2018 Tenth International Conference on Advanced Computing (ICoAC)*. 2018, s. 88–93.

[53] Reis, D. a Mader, G. *Odoo 15 Development Essentials: Enhance your Odoo development skills to create powerful business applications*. 1. vyd. Packt Publishing, 2022. ISBN 9781800203082. Dostupné z: https://books.google.sk/books?id=QJlXEAAAQBAJ.

[54] Rychlík, J. *Rozděleni Československa 1989-1992*. Vyšehrad, 2022. ISBN 978-80-7601-603-3.

[55] Scott, A. W. *Mapping objects to relational databases: O/R mapping in detail*. 2006. Dostupné z: https://agiledata.org/essays/mappingObjects.html.

[56] Národná rada Slovenskej republiky. Zákon o samospráve vyšších územných celkov (zákon o samosprávnych krajoch). In: *Zbierka zákonov Slovenskej republiky*. Ministerstvo spravodlivosti Slovenskej republiky, 2001, čiastka 125/2001, s. 1–17. ISSN 2644-4674.

[57] Snášel, R. *Snesení Zastupitelstva a Rady Obce*. 2014. Dostupné z: https://www.uzob.cz/.

[58] Štěpán, S. *Transformace systému z monolitické architektury do architektury mikroslužeb*. 2021. Diplomová práca. České vysoké učení technické v Praze. Vypočetní a informační centrum. Dostupné z: https://dspace.cvut.cz/handle/10467/94558.

[59] Taibi, D.; Lenarduzzi, V.; Pahl, C. a Janes, A. Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In: *Proceedings of the XP2017 Scientific Workshops*. New York, NY, USA: Association for Computing Machinery, 2017. XP '17. ISBN 9781450352642. Dostupné z: https://doi.org/10.1145/3120459.3120483.

[60] Tan, W.; Fan, Y.; Ghoneim, A.; Hossain, M. A. a Dustdar, S. From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing*, 2016, zv. 20, č. 4.

[61] the SQLAlchemy authors and contributors. *SQLAlchemy 2.0 Documentation*. 2024. Dostupné z: https://docs.sqlalchemy.org/en/20/errors.html#type-annotation-can-t-be-interpreted-for-annotated-declarative-table-form.

[62] Vasista, T. G. K. a AlSudairi, M. A. T. Service-Oriented Architecture (SOA) and Semantic Web Services for Web Portal Integration. In: Meghanathan, N.; Nagamalai, D. a Chaki, N., ed. *Advances in Computing and Information Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 253–261. ISBN 978-3-642-31552-7.

[63] Vasudevan, K. *Swagger*. 2023. Dostupné z: https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/.

[64] Wang, Y.-H. a Liao, J. C. Why or Why Not Service Oriented Architecture. In: *2009 IITA International Conference on Services Science, Management and Engineering*. 2009, s. 65–68.

[65] Wikipédia. *Administratívne členenie Slovenska — Wikipédia, Slobodná encyklopédia*. 2023. Dostupné z: //sk.wikipedia.org/w/index.php?title=Administrat%C3%ADvne_%C4%8Dlenenie_Slovenska&oldid=7686138.

[66] Wikipédia. *Česko — Wikipédia, Slobodná encyklopédia*. 2024. Dostupné z: //sk.wikipedia.org/w/index.php?title=%C4%8Cesko&oldid=7956342.

[67] Zaklová, K.; Hynek, J. a Hruška, T. Towards Transparent Governance: Unifying City Councils Decision-Making Data Processing and Visualization. In: Springer. *World Conference on Information Systems and Technologies*. 2024, s. 402–411. ISBN 978-3-031-60221-4.

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

[69] Zyl, P. van; Kourie, D. G.; Coetzee, L. a Boake, A. The influence of optimisations on the performance of an object relational mapping tool. In: Library, A. D., ed. *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. New York, NY, USA: Association for Computing Machinery, 2009, s. 150–159. SAICSIT '09. ISBN 9781605586434. Dostupné z: https://doi.org/10.1145/1632149.1632169.

[70] Šimková, P.; Slavíček, P.; Kadlecová, K.; Urbánková, A.; Krumlová, J. et al. *Metodické doporučení k činnosti územních samosprávných celků: č. 9– Nejčastější nedostatky při výkonu samostatné působnosti obcí*. Ministerstvo vnitra České republiky, odbor dozoru a kontroly veřejné správy, 2013. Dostupné z: https://www.mvcr.cz/odk2/clanek/metodicke-materialy-k-zakonnym-zmocnenim.aspx.

Príloha A

Koncové body REST API

Inicializačné koncové body

- `/flask/hello`
  Inicializačný koncový bod. Ako jediný nekomunikuje s databázou. Pre overenie funkčnosti serveru.
- `/flask/appInfo`
  Základné informácie o aplikácii Zastupko.cz.

Koncové body pre všeobecné dáta k zastupitelstvu

- `/flask/<municipality_id>/<body>/<int:council_id>/dataset`
  Koncový bod poskytujúci celú dátovú sadu pre zvolené zastupiteľstvo, tzn. za jedno funkčné obdobie.
- `/flask/<municipality_id>/<body>/<int:council_id>/dataset`
  Nový koncový bod poskytujúci vyhladávanie nad členmi, hlasovaniami a politickými stranama. Prijima volitelný parameter `search`
- `/flask/<municipality_id>/<body>/<int:council_id>/generalData`
  Základné dáta o konkrétnom zastupiteľstve (primárne pre úvodný dashboard so súhrnnými štatistikami).
- `/flask/<municipality_id>/<body>/<int:council_id>/member/<int:member_id>`
  Dáta o členovi/členkyni zastupiteľstva.

Koncové body pre špecifické dáta municipality a jej metadáta

- `/flask/<municipality_id>/<body>/councils`
  Zoznam dostupných zastupiteľstiev z dané municipality.
- `/flask/municipalities`
  Zoznam municipalít.
- `/flask/municipality/<municipality_id>`
  Dáta o municipalite a jej orgánoch.

- `/flask/municipality/<municipality_id>/logo`
  Pre získanie loga municipality.

Koncové body pre dáta o zasadaniach

- `/flask/<municipality_id>/<body>/<int:council_id>/sessionsData`
  Základné dáta o všetkých zasadaniach konkrétneho orgánu municipality.
- `/flask/<municipality_id>/<body>/<int:council_id>/session/<int:session_id>`
  Dáta o jednom zasadaní orgánu.
- `/flask/<municipality_id>/<body>/<int:council_id>/sessionAttendance/<int:session_id>`
  Dáta k dochádzke z jedného zasadania orgánu.

Koncové body pre dáta o hlasovaniach

- `/flask/<municipality_id>/<body>/<int:council_id>/votesData`
  Základné dáta o všetkých hlasovaniach konkrétneho zastupiteľstva.
- `/flask/<municipality_id>/<body>/<int:council_id>/votes`
  Koncový bod pre filtrovanie hlasovaní.
- `/flask/<municipality_id>/<body>/<int:council_id>/vote/<int:vote_id>`
  Dáta o jednom hlasovaní zastupiteľstva.

Koncové body pre dáta o politických subjektoch

- `/flask/<municipality_id>/<body>/<int:council_id>/partiesData`
  Základné dáta o všetkých politických subjektoch konkrétneho zastupiteľstva.
- `/flask/<municipality_id>/<body>/<int:council_id>/partiesStats`
  Štatistika hlasovania jednotlivých politických subjektov zastupiteľstva.
- `/flask/<municipality_id>/<body>/<int:council_id>/party/<int:party_id>`
  Dáta o jednom politickom subjekte zastupiteľstva.

Koncové body pre FE komponenty

- `/flask/FE/<municipality_id>/<body>/<int:council_id>/dropdownMemberOptions`
  Zoznam členov zastupiteľstva do komponenty `Dropdown`.
- `/flask/FE/<municipality_id>/<body>/<int:council_id>/dropdownSessionOptions`
  Zoznam zasadaní zastupiteľstva do komponenty `Dropdown`.
- `/flask/FE/<municipality_id>/<body>/dropdownSessionResults`
  Obsah číselníka možností výsledkov hlasovaní do komponenty `Dropdown`.

- `/flask/FE/<municipality_id>/<body>/<int:council_id>/dropdownSubjectOptions`
  Zoznam politických subjektov zastupiteľstva do komponenty `Dropdown`.
- `/flask/FE/<municipality_id>/<body>/dropdownVoteOptions`
  Obsah číselníka základných hlasovacích možností do komponenty `Dropdown`.
- `/flask/FE/<municipality_id>/<body>/dropdownOtherVoteOptions/<int:member_id>`
  Obsah číselníka ostatných hlasovacích možností zastupiteľa do komponenty `Dropdown`.
- `/flask/FE/<municipality_id>/<body>/<int:council_id>/dropdownYearOptions`
  Zoznam jednotlivých rokov vo funkčnom období zastupiteľstva do komponenty `Dropdown`.

Koncové body pre pokročilé analýzy

- `/flask/<municipality_id>/<body>/<int:council_id>/attendanceDashboard`
  Dáta pre analýzu dochádzky zastupiteľov.
- `/flask/<municipality_id>/<body>/<int:council_id>/compare`
  Koncový bod pre porovnanie dvoch členov zastupiteľstva.
- `/flask/<municipality_id>/<body>/<int:council_id>/PartiesUnity`
  Koncový bod pre dáta zobrazované v grafe nejednotnosti strán.

Príloha B

Návrh adresárovej štruktúry

V nasledujúcich prílohách je zobrazená nová adresárová štruktúra serverovej časti. Vzhľadom na jej rozmanitosť je rozdelená na tri časti, kde obrázok B.1 popisuje koreňovú štruktúru adresárov, obrázok B.2 štruktúru súborov verejného modulu a B.3 rozmiestnenie tabuliek databáz.

```
api/ ..... koreňový adresár
├── admin/ ..... zdrojové súbory pre administračný modul
├── api_test/ ..... testy serverovej časti
├── migrations/ ..... skripty migrácie databáz
├── loga/ ..... adresár s logami municipalít
├── orm/ ..... zdrojové súbory pre objektovo relačné mapovanie
├── public/ ..... verejná časť
│   ├── resources/ ..... triedy koncových bodov
│   ├── service/ ..... volania databázy
│   └── util/ ..... pomocné triedy a funkcie
├── create_app.py/ ..... inicializačný súbor API
├── blueprints.py/ ..... súbor v ktorom sú definované blueprints
├── error_handlers.py/ ..... definície chybových funkcií pre moduly serverovej časti
├── app.py/ ..... štartovaci bod aplikácie
└── config.py/ ..... konfiguračný súbor
```

Obr. B.1: Obsah koreňovej adresárovej štruktúry serverovej časti. Modrou farbou označené zložky/súbory sú vyznačené zmeny v adresárovej štruktúre.

Obr. B.2: Návrh súborovej štruktúry novej architektúry vo verejnom module. Modrou farbou označené zložky/súbory sú vyznačené zmeny v adresárovej štruktúre.

Obr. B.3: Návrh adresárovej štruktúry rozmiestnenia tabuliek databázy. Modrou farbou označené zložky/súbory sú vyznačené zmeny v adresárovej štruktúre.