VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

WEBOVÁ APLIKACE PRO PLÁNOVÁNÍ ROZMÍSTĚNÍ SBĚRNÝCH MÍST ODPADU
WEB APPLICATION FOR WASTE CONTAINERS PLACEMENT PLANNING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Adam Vidlička**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ladislav Dobrovský**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Adam Vidlička**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Ladislav Dobrovský**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Webová aplikace pro plánování rozmístění sběrných míst odpadu

Stručná charakteristika problematiky úkolu:

V posledních letech byla vyvinuta řada algoritmů pro různé aplikace v problematice nakládání s odpadem, ať jde o rozmístění kontejnerů, výstavbu zařízení na zpracování odpadu, nebo nalezení optimálních tras pro svozová vozidla. Dalším krokem je převedení těchto výsledků do praxe. Moderní webová aplikace by měla splňovat určité požadavky na responsivitu a oddělení kódu na klientovi a serveru pomocí zasílání asynchronních požadavků (AJAX) nebo přímo aktivním duplexním spojením (WebSockets). Aplikace bude postavena na reálných datech a jednotlivé prvky aplikace budou konzultovány s odborníky z Ústavu procesního inženýrství.

Cíle diplomové práce:

Rešerše využití poskytnutého backendu (Flask) a rozšíření o relevantní endpointy a moduly.
Rešerše a zvolení vhodných frontend technologií v kooperaci s dalším studentem.
Vytvoření webové aplikace, která bude soužit uživatelům (jednotlivým obcím, svazům obcí, či firmám) při plánování umístění sběrných míst v odpadovém hospodářství:
Návrh schematu databáze s ER diagramem, jak společné části aplikace, tak specifické pro problém umisťování sběrných míst.
Mockupy pro vstupní i výstupní API.
Propojení s externím výpočetním jádrem.
Uživatelské rozhraní orientované na práci s myší či dotykovou obrazovkou většího formátu (monitor, větší tablet).
Práce s mapovými podklady.

Seznam doporučené literatury:

CHATURVEDI, Rajneesh, Swati V. CHANDE a Amita SHARMA. Evaluation and Refinement of MVC Web Application Architecture. Journal of Information and Computational Science. 2021, 2021(3). ISSN 1548-7741.

HAKLAY, M. Openstreetmap: User-generated street maps. IEEE Pervasive Computing [online]. 2008, 7(4), 12 [cit. 2021-10-21]. ISSN 1536-1268.

W3 schools: AJAX Introduction [online]. [cit. 2021-10-21]. Dostupné z: https://www.w3schools.com/xml/ajax_intro.asp.

OpenStreetMap documentation: Deploying your own Slippy Map [online]. [cit. 2021-10-21]. Dostupné z: https://wiki.openstreetmap.org/wiki/Deploying_your_own_Slippy_Map.

Flask-SocketIO [online]. [cit. 2021-10-21]. Dostupné z: https://flask-socketio.readthedocs.io/.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Cílem této diplomové práce je vytvoření webové aplikace, která bude sloužit jako nástroj pro vytváření a správu sběrné infrastruktury obcí. K optimálnějšímu vytváření sběrné infrastruktury se využijí výpočetní funkce, které se vyvíjejí na Ústavu procesního inženýrství. Teoretická část práce se zabývá přiblížením odpadového hospodářství v České republice, motivací vzniku webové aplikace a použitými technologiemi při vývoji webové aplikace. V praktické části práce je popis vzniku základního konceptu webové aplikace a následný popis jednotlivých částí webové aplikace.

ABSTRACT

The aim of this master's thesis is to create a web application that will serve as a tool for creation and management of municipal waste collection infrastructure. Computational functions developed at the Institute of Process Engineering will be utilized for more optimal creation of the waste collection infrastructure. The theoretical part of the thesis deals with the zoom in on waste management in the Czech Republic, the motivation behind developing the web application, and the technologies used during its development. The practical part of the thesis describes the creation of the basic concept of the web application and provides subsequent descriptions of its individual components.

KLÍČOVÁ SLOVA

Webová aplikace, odpadové hospodářství, sběr odpadu, sběrný systém, React, Flask, Redux Toolkit

KEYWORDS

Web application, waste management, waste collection, collection system, React, Flask, Redux Toolkit

ÚSTAV AUTOMATIZACE A INFORMATIKY

2023

BIBLIOGRAFICKÁ CITACE

VIDLIČKA, Adam. *Webová aplikace pro plánování rozmístění sběrných míst odpadu*. Brno, 2023. Dostupné také z: `https://www.vut.cz/studenti/zav-prace/detail/145895`. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Vedoucí práce: Ing. Ladislav Dobrovský

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 26. 5. 2023 .....

Adam Vidlička

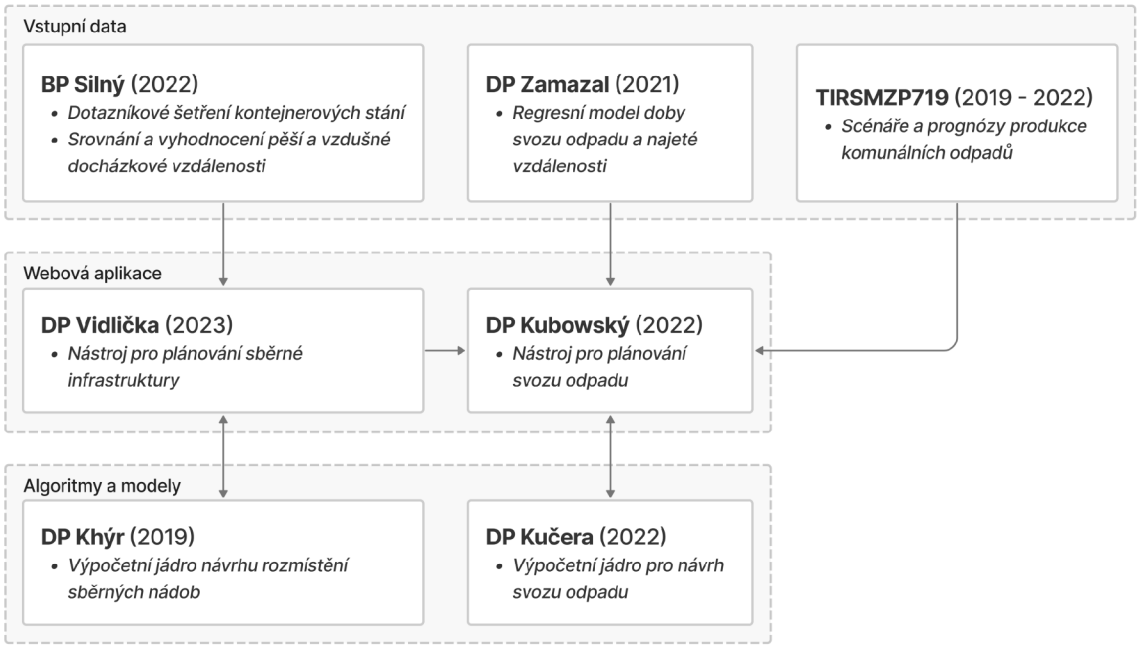
PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Ladislavu Dobrovskému za odborné vedení a cenné rady při tvorbě této práce. Dále děkuji procovníkům z Ústavu procesního inženýrství Ing. Vlastimírovi Nevrlému, Ph.D., Ing. Radovanovi Šomplákovi, Ph.D. a Ing. Veronice Smejkalové, Ph.D. za spolupráci a konzultace k postupu vývoje webové aplikace. Děkuji také Ing. Jiřímu Kubowskému za spolupráci při návrhu a vývoji webové aplikace. Na závěr bych chtěl poděkovat i své rodině za podporu během studia.

OBSAH

1 ÚVOD

Tato diplomová práce je vedena ve spolupráci s Ústavem procesního inženýrství (ÚPI), na kterém jsou vyvíjeny modely a nástroje pro plánování infrastruktury v odpadovém hospodářství. Praktické využití těchto modelů a nástrojů je často limitováno z důvodu nutného softwarového vybavení, které klade nemalé nároky také na samotné uživatele. Vznikl proto koncept závěrečných prací na ÚPI, Ústavu matematiky a Ústavu automatizace a informatiky, do kterého zapadá i tato diplomová práce. Cílem je vytvořit komplexní nástroj pojmenovaný Popelka pro plánování sběrné a svozové infrastruktury v odpadovém hospodářství. Propojení závěrečných prací a výzkumných projektů zapojených do projektu Popelka je znázorněno na obr. 1. Výsledkem spolupráce bude nástroj v podobě webové aplikace, která bude sdružovat více modelů a výsledky budou vizualizovány pomocí mapových podkladů, grafů a tabulek. Vznikne tak uživatelsky přívětivé prostředí, které umožní na základě nastavených vstupních parametrů vyhodnotit vhodnou infrastrukturu odpadového hospodářství.



Obr. 1: Schéma projektu Popelka.

Výsledky jednotlivých částí (viz. obr. 1) a dříve zmíněných studií budou sloužit jako vstup a výpočtové jádro ve vyvíjené aplikaci. Výsledky alokace sběrných nádob mohou být dále modifikovány např. starostou, který bude měnit umístění s ohledem na lokální podmínky a možnosti (existence vyhrazených a ohraničených kontejnerových stání, estetika okolního prostředí, návaznost na dopravní infrastrukturu apod.). V rámci obsluhy aplikace by měly být implementovány i další funk-

cionality související s reálným počtem obyvatel, úpravy kapacit sběrné soustavy s ohledem na produkci daného typu odpadu a další. Webová aplikace by měla umět volat výpočetní jádro a definovat okrajové podmínky úlohy – docházkovou vzdálenost ke sběrným nádobám, počet sběrných míst, příp. celkové pořizovací náklady na sběrné nádoby. Pro výpočet bude využit výpočetní nástroj vyvíjený týmem na ÚPI [1, 2]. Uživatel po zadání vstupních parametrů dostane návrhy optimálního rozmístění nádob pro scénář odpovídající řešené úloze v dané obci. Výsledky budou ukládány v rámci databáze a budou zobrazovány v mapových podkladech, kde budou znázorňovat navržené rozmístění nádob v obci nebo části obce. S ohledem na vývoj územního plánování bude aplikace disponovat možností přidávání a odebírání uzlů popisujících produkční a sběrná místa. Předpokládá se, že databáze bude obsahovat všechny obce ČR.

Paralelně s návrhem aplikace a funkcionalit probíhalo dotazníkové šetření [3], které definuje kritéria pro úlohu matematického programování, na jejímž místě bude úloha optimalizace sběrných míst upravena [1, 2]. Dalším vstupem do aplikace je množina scénářů produkce pro různé frakce komunálního odpadu. Tento vstup vychází z výsledků několikaletého projektu pro Ministerstvo životního prostředí [4]. Modul pro plánování sběrné infrastruktury je prerekvizitou pro návazný modul sloužící k plánování svozu odpadu, který je vytvářen v rámci paralelně řešené závěrečné práce [5].

Cílem aplikace je vytvořit podpůrné podklady pro ekonomicky a environmentálně udržitelný systém odpadového hospodářství v obcích. Návrh nové podoby infrastruktury bude především podporovat separaci odpadu při současné ekonomické udržitelnosti systému a to např. v podobě meziobecní spolupráce. Výsledná aplikace a databáze budou podporovat dlouhodobou udržitelnost, tedy budou umožňovat nastavovat okrajové podmínky na míru regionu a obci. Výstupy budou zaměřeny jak na vybrané frakce odpadu, které se již samostatně sbírají, tak i na odpady, které se ještě ve většině obcí neseparují. Příkladem frakce odpadu, kterou prozatím všechny obce neseparují, je textil s povinností separace od roku 2025 dle zákonu č. 541/2020 Sb., o odpadech. Jedná se tedy o podpůrný nástroj pro obce, který umožní efektivně navrhnout sběrnou soustavu.

2 ODPADOVÉ HOSPODÁŘSTVÍ

Odpadové hospodářství je založeno na hierarchii odpadového hospodářství, podle níž je prioritou předcházení vzniku odpadu, a nelze-li vzniku odpadu předejít, pak v následujícím pořadí jeho příprava k opětovnému použití, recyklace, jiné využití, včetně energetického využití, a není-li možné ani to, jeho odstranění. [6]

V České republice vznikl první zákon o odpadech v roce 1991. V současnosti nakládání s odpady upravuje zákon č. 541/2020 Sb., o odpadech, který je účinný od 1. 1. 2021. Zákon stanovuje práva a povinnosti osobám v oblasti odpadového hospodářství a prosazuje základní principy oběhového hospodářství[1], ochrany životního prostředí a zdraví lidí při nakládání s odpady. Nakládání s výrobky s ukončenou životností (viz. kap. 2.2) upravuje zákon č. 542/2020 Sb., o výrobcích s ukončenou životností, účinný od 1. 1. 2021. Nakládání s odpady z obalů upravuje zákon č. 477/2001 Sb., o obalech, ve znění pozdějších předpisů. [6]

Odpady vznikají prakticky při veškeré lidské činnosti. Vnikají v průmyslu, stavebnictví, zemědělství, dopravě a při běžném životě člověka. Zejména komunální odpady jsou produktem všech obyvatel. Kvůli svým specifickým vlastnostem a různému riziku ohrožení životního prostředí vyžaduje každý tok odpadů specifické nakládání. Základní pravidla pro nakládání s odpady jsou stanovena zákonem č. 541/2020 Sb., o odpadech a jeho prováděcími právními předpisy. Cílem pro nakládání s odpady a opatření pro jejich dosažení jsou stanoveny Plánem odpadového hospodářství České republiky (POH ČR). Jeho plnění je vyhodnocováno prostřednictvím hodnotících zpráv [8]. S plánem odpadového hospodářství ČR musí být v souladu také plány odpadového hospodářství krajů. [9]

Za účelem pravidelného vyhodnocení odpadového hospodářství, a pro získání podkladů pro správní a kontrolní činnost, je v odpadovém hospodářství vedena evidence odpadů, umožňující v souladu s evropskými předpisy získat podrobné informace o produkci a nakládání s odpady. Získané informace jsou důležitým podkladem pro další strategické plánování v oblasti odpadového hospodářství, oběhového hospodářství a legislativní činnost Ministerstva životního prostředí. Oblast nakládání s odpady zahrnuje také přeshraniční přepravu odpadů z ČR a do ČR či přes její hranice. Přeshraniční přeprava je upravena právními předpisy EU a je povolována v rámci správního řízení tak, aby byly minimalizovány její rizika a dopady na životní prostředí. [9]

[1] Oběhové hospodářství (cirkulární ekonomika) si klade za cíl udržet hodnotu výrobků, materiálů a zdrojů tak dlouho v ekonomickém cyklu, jak je to jen možné, a vrátit je do výrobního cyklu na konci jejich životnosti, přičemž se minimalizuje tvorba odpadu. Oběhové hospodářství se stalo jedním z klíčových konceptů v oblasti řady politik Evropské unie. [7]

Pro splnění cílů ohledně nakládání s komunálním odpadem na úrovni státu je nezbytné realizovat akční kroky již na úrovni obcí a podpořit tak žádoucí způsoby nakládání s odpady. Navýšení míry separace komunálního odpadu, což je nutný předpoklad pro následnou recyklaci, je důsledkem činnosti každého občana. Zavádí se motivační systémy pro obce a domácnosti, aby s odpady nakládaly způsoby, které jsou definované v obecně závazných vyhláškách (předcházely vzniku odpadu, využitelné složky sbíraly odděleně na místech určených obcí).

2.1 Plán odpadového hospodářství ČR

POH ČR pro období 2015-2024 byl schválen vládou 22. 12. 2014. Plán odpadového hospodářství České republiky je nástroj pro řízení odpadového hospodářství ČR a pro realizaci dlouhodobé strategie odpadového hospodářství. Povinnost ČR zpracovat plán nakládání s odpady na jejím území je stanovena ve směrnici Evropského parlamentu a Rady 2008/98/ES o odpadech. Ministerstvo životního prostředí podle zákona o odpadech zpracovalo POH ČR ve spolupráci s příslušnými orgány veřejné správy a veřejností.

11. května 2022 vláda ČR schválila aktualizaci POH ČR s výhledem do roku 2035. ČR má aktualizovanou hlavní strategii odpadového hospodářství. V POH ČR jsou zakomponovány veškeré cíle novelizovaných evropských směrnic, nového zákona o odpadech, zákona o výrobcích s ukončenou životností a novely zákona o obalech.

Plán představuje klíčový dokument pro realizaci dlouhodobé strategie nakládání s odpady, obalovými odpady a výrobky s ukončenou životností. Hlavními cíli strategie je jednoznačně přechod k oběhovému hospodářství, předcházení vzniku odpadů, zvýšení recyklace a materiálového využití odpadů. Součástí POH ČR je i Program předcházení vzniku odpadů.

Plán se zaměřuje na upřednostnění způsobů nakládání s odpady podle hierarchie odpadového hospodářství a plnění evropských cílů ve všech oblastech nakládání s odpady. Strategie navržená v POH ČR vede k odklonu odpadů ze skládek skrze předcházení odpadů, zvýšení recyklace a materiálové využití odpadů. Strategické cíle uvedené v POH ČR jsou:

1. Předcházení vzniku odpadů a snižování měrné produkce odpadů.
2. Minimalizace nepříznivých účinků vzniku odpadů a nakládání s nimi na lidské zdraví a životní prostředí.
3. Udržitelný rozvoj společnosti a přechod k cirkulární ekonomice.
4. Maximální využívání odpadů jako náhrady primárních zdrojů.

Z priorit POH ČR vyplývá i nezbytnost stanovit a koordinovat síť zařízení k nakládání s odpady v regionech. Na POH ČR tak přímo navazuje nový programový

dokument Operačního programu Životní prostředí, prostřednictvím kterého je možné čerpat finance pro podporu nových zařízení a systémů nakládání s odpady v ČR.

POH ČR jako strategický rámec pro rozvoj nakládání s odpady je plně v souladu s evropskou odpadovou legislativou. POH ČR je určujícím dokumentem pro tvorbu plánů odpadového hospodářství jednotlivých krajů. [10]

2.2 Zpětný odběr výrobků

Zpětný odběr výrobků (ZOV) vychází z principu individuální odpovědnosti výrobce zajistit nakládání s výrobky po ukončení jejich životnosti. Smyslem ZOV je motivovat výrobce k navrhování a produkci výrobků s co možná nejnižším obsahem nebezpečných látek, jejichž následné využití nebo odstranění po ukončení životnosti bude co nejlevnější a nejjednodušší. Z tohoto hlediska je tedy žádoucí, aby bylo možné výrobky co nejčastěji opětovně používat, recyklovat je a minimalizovat vznik odpadu z jejich zpracování.

Při plnění cílů stanovených pro ZOV po ukončení jejich životnosti hrají klíčovou roli koneční uživatelé výrobků, kteří musí být informováni jak a kde lze výrobky s ukončenou životností odevzdat, a kteří budou motivováni k tomu, aby se daných výrobků nezbavovali v rámci směsného komunálního odpadu (SKO).

Odpovědnost za celý životní cyklus výrobku včetně zajištění ZOV je stanovena v souladu s evropskou legislativou všem osobám, které uvádějí na trh v ČR obaly, vozidla, elektrická a elektronická zařízení, baterie a akumulátory a v neposlední řadě pneumatiky. Legislativně je oblast ZOV upravena zákonem o obalech a zákonem o výrobcích s ukončenou životností, a navazujícími prováděcími právními předpisy.

V minulosti byl zaveden systém zpětného odběru minerálních olejů, avšak od 1. října 2015 byly oleje z vybraných výrobků vyřazeny, a od tohoto data se na oleje již zpětný odběr nevztahuje. S použitými oleji již není možné nakládat v režimu zpětného odběru, ale pouze v režimu nebezpečných odpadů. [11]

2.3 Sběr odpadu

Volba a návrh sběrné infrastruktury by měly být prováděny pro každý typ odpadu, jehož zpracování je regulováno v Balíčku oběhového hospodářství [12]. Infrastruktura sběrných míst v dané lokalitě musí být dimenzována na svoz a finální zpracování veškerého produkovaného odpadu. Zásadním vstupem pro strategické plánování je očekávaná produkce odpadu v budoucnu. Jedním z prvních kroků při vytváření infrastruktury pro sběr odpadu je rozmístění sběrných nádob. Tato problematika je často založená na geografickém informačním systému [13] spojeném s optimalizací

využívající data z map. Dalším přístupem je optimalizace z pohledu jiných kritérií. Hemmelmayr et al. [14] představili model pro umístění nádob se současným plánováním sběru odpadu, přičemž byly minimalizovány celkové náklady. Rossit et al. [15] cílili na vývoj vícekriteriálního modelu, kde byla řešena frekvence sběru s ohledem na kapacitu sběrných nádob, průměrnou docházkovou vzdálenost a náklady na pořízení nádob. Poskytnutý model představuje kvalitní základ pro rozhodování v oblasti nakládání s odpady. Možnost dalšího rozvoje byla nalezena v oblasti vazeb kritérií a jejich kombinací v rámci víceúčelové formulace, této problematice se věnovali Nevrlý et al. [2]. Závěrem studie je, že lokace sběrných nádob by měla být ekonomicky přijatelná a současně občanům poskytovat komfort z pohledu docházkové vzdálenosti ke sběrným místům.

3 POUŽITÉ TECHNOLOGIE

V této kapitole jsou popsány použité technologie pro vytvoření webové aplikace. Bylo potřeba vybrat vhodnou kombinaci technologií, aby bylo možné splnit základní požadavky od aplikace a následně bylo i jednoduché její rozšíření dle nových požadavků od uživatelů, kteří aplikaci budou testovat a používat.

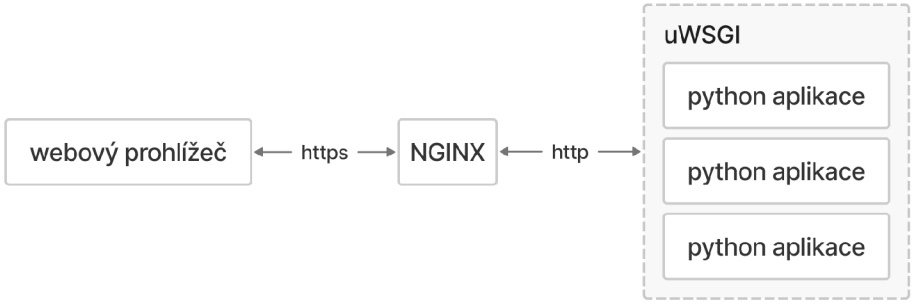
Aplikace má základní rozdělení na serverovou část (backend) a klientskou část (frontend). Pro každou z těchto částí bylo možné zvolit jiný programovací jazyk. Pro backend byl zvolen programovací jazyk Python a pro frontend byl zvolen jazyk JavaScript v kombinaci s HTML a CSS.

3.1 Technologie na backendu

Pro vytvoření backendu aplikace byla možnost využití řešení backendu od vedoucího diplomové práce. Rozhodlo se, že se toto řešení využije a z jeho částí se vytvoří nové jádro backendu pro aplikaci, na které se naváží další moduly.

Backend je napsán v programovacím jazyku Python za využití frameworku Flask (viz. kap. 3.1.1), který sice nenabízí komplexní řešení jako např. Python framework Django, ale je za to jednodušší a dává větší svobodu ve volbě knihoven pro aktuální problematiku.

Součástí jádra backendu je také využití komunikace s databází MariaDB (viz. kap. 3.1.2) a využití služby NGINX (viz. kap. 3.1.3), která zpřístupňuje aplikaci uživatelům (komunikace s aplikací probíhá přes socket) a přidává SSL/TLS šifrování komunikace. V kombinaci s NGINX se využívá služba uWSGI (viz. kap. 3.1.4), která zajišťuje spuštění více instancí aplikace, mezi které se rozkládají uživatelské požadavky. Tato komunikace je zobrazena na obr. 2.



Obr. 2: Komunikace webového prohlížeče, NGINX a uWSGI.

Rozšířením jádra backendu bylo přidání Redis (viz. kap. 3.1.5) databáze pro ukládání uživatelských a dočasných dat. V souvislosti s tím vznikla možnost vytvoření fronty úloh Redis Queue (viz. kap. 3.1.6) pro výpočty.

3.1.1 Flask

Flask je open source[1] webový framework napsaný v jazyce Python a slouží k vytváření webových aplikací. Je postavený na knihovně Werkzeug, což je WSGI knihovna, která poskytuje způsob, jak zpracovávat HTTP požadavky a odpovědi. [16, 17]

Jedná se o mikroframework, což znamená, že nabízí minimální a jednoduchý základ, který se dá dále rozšiřovat o další knihovny dle potřeb vývojáře. Flask nabízí základní funkce pro tvorbu webových aplikací jako např. routing, HTML šablony pomocí Jinja template engine, správa HTTP požadavků a odpovědí, práce s databázemi a další. Flask tedy nenabízí komplexní řešení již od začátku, což může být pro vytvářenou aplikaci a vývojáře výhodou, protože je volnost ve výběru knihoven, které se nejvíce hodí pro řešení daného problému.

3.1.2 MariaDB

MariaDB je relační databázový systém, který vychází z MySQL a stojí za ním vývojáři MySQL. MariaDB je tedy kompatibilní s MySQL. Jedná se o open source software, který poskytuje výkonnost, stabilitu a otevřenost, což znamená podporu komunitou. [18]

Relační databáze se skládají z tabulek, které obsahují řádky a sloupce. Řádek představuje záznam/položku jejíž atributy jsou definované pomocí sloupců a zároveň obsahuje sloupec s unikátním id nazývaným klíč. Vazby mezi tabulkami jsou vytvářeny pomocí klíčů. Tabulky a vazby mezi tabulkami nabízejí strukturovanost. Pro manipulaci s tabulkami slouží standardizovaný programovací jazyk SQL, který nabízí sadu příkazů. Mezi nejčastěji používané příkazy pro manipulaci s daty v tabulkách patří:

- SELECT – příkaz slouží k vyhledávání a získávání dat ze zvolené tabulky v databázi. V rámci něj lze specifikovat požadované sloupce, které chceme získat, nebo také podmínky, na základě kterých mají být záznamy získány. Příkaz umožňuje i spojování tabulek pomocí klíčů nebo na základě jiných společných hodnot, takže je možná vybrat i sloupce z jiných tabulek v rámci jednoho dotazu.
- INSERT – příkaz slouží k vkládání nových záznamů dat do databáze. Je potřeba specifikovat tabulku, do které se má provést zápis, a hodnoty, které mají být vloženy.
- UPDATE – příkaz slouží k aktualizování existujících záznamů v databázi. V rámci příkazu se specifikuje tabulka a hodnota, která se má aktualizovat.

[1] Open source je označení softwaru, jehož zdrojový kód je veřejně dostupný, svobodně použitelný s možností modifikace a distribuce.

Pokud je potřeba aktualizovat pouze část záznamů v tabulce, lze využít podmínku, po jejímž splnění se tyto záznamy aktualizují.

- DELETE – příkaz slouží k mazání dat z databáze. Je zapotřebí specifikovat tabulku, ve které se má provést mazání. Pokud se mazání týká pouze některých záznamů v tabulce, je potřeba je odfiltrovat pomocí podmínky.

3.1.3 NGINX

NGINX je open source software pro poskytování webu, reverzní proxy, poskytování mezipaměti, vyvažování zátěže a dalších. Ze začátku to byl webový server navrhovaný pro maximální výkon a stabilitu. Kromě schopnosti HTTP serveru, může NGINX fungovat také jako proxy server pro e-maily (IMAP, POP3 a SMTP), reverzní proxy a vyvažovač zátěže pro HTTP, TCP a UDP servery. [19]

V rámci vyvíjené webové aplikace, NGINX zpřístupňuje aplikaci uživatelům (komunikace s aplikací probíhá přes socket) a přidává SSL/TLS šifrování komunikace.

3.1.4 uWSGI

uWSGI je open source software, který se zaměřuje na vývoj kompletního stacku pro vytváření hostingových služeb. Aplikační server (pro různé programovací jazyky a protokoly), proxy, správce procesů a monitorování jsou všechny implementovány pomocí společného rozhraní API a společného konfiguračního stylu. [20]

V rámci vyvíjené webové aplikace, uWSGI zajišťuje spuštění více instancí aplikace, mezi které se rozkládají uživatelské požadavky.

3.1.5 Redis

Redis je open source úložiště dat v paměti a využívá se jako databáze, mezipaměť (pro databázové dotazy, komplexní výpočty, volání API a ukládání relací), engine pro streamování a zprostředkovatel zpráv. Podporuje datové struktury jako např. string, hash, list, set a další. Data jsou držena v paměti pro rychlý přístup, ale je i možnost jejich perzistence (data se neztratí po restartu nebo selhání systému) pomocí zápisu do permanentního úložiště (uložení na disk). [21]

3.1.6 Redis Queue

Redis Queue je jednoduchá Python knihovna pro řazení úloh do fronty a jejich zpracování na pozadí za pomocí workerů. Je podporovaná Redisem a je navržena tak, aby měla nízkou bariéru vstupu a šlo ji jednoduše integrovat do projektu. [22]

3.2 Technologie na frontendu

V rámci poskytnutého backendu bylo i řešení frontendové části a to za využití Brythonu (Python knihovna). Rozhodlo se toto řešení nahradit kombinací HTML, CSS a JavaScriptu a to převážně z důvodu větší zkušenosti studentů s JavaScriptem oproti Pythonu. Přechodem na JavaScript se také otevřelo několik možností využití knihoven (např. React) nebo frameworků (např. Angular), které jsou na něm postaveny.

Obr. 3: Aplikace s použitím (a) frameworku vs. (b) knihoven.

Pro aplikaci se zvolilo využití knihoven. Hlavním důvodem byla počáteční domluva, že se využije system Flasku, který bude odesílat HTML soubory na klienta. Z tohoto důvodu se ustoupilo od použití Node.js jako JavaScript runtime environment (prostředí pro spouštění JavaScriptu) a vyřadilo se použití frameworků, kdy většina z nich potřebuje build proces[2] pomocí Node.js. Dalším důvodem byl rozdíl mezi knihovnou a frameworkem (viz. obr. 3). Framework představuje komplexní řešení, které má svá pravidla, která je potřeba dodržovat. Naopak knihovna dává volnost v tom, kdy a jak se použije, což umožňuje postupné rozšiřování projektu o potřebné knihovny. Tedy projekt se staví na frameworku a přizpůsobuje se jeho pravidlům, ale naopak knihovna staví na projektu a programátor má větší kontrolu nad použitými knihovnami.

Základ frontendu je postavený na využití knihovny React (viz. kap. 3.2.1) a routing (směrování) v rámci aplikace, tedy mezi jednotlivými moduly, je zařízeno pomocí knihovny React Router (viz. kap. 3.2.3). Další velkou částí aplikace je zobrazování mapových podkladů, které je řešeno využitím knihoven Leaflet a React Leaflet (viz. kap. 3.2.4). Zobrazování statistických dat je vytvořeno pomocí knihovny Recharts (viz. kap. 3.2.6).

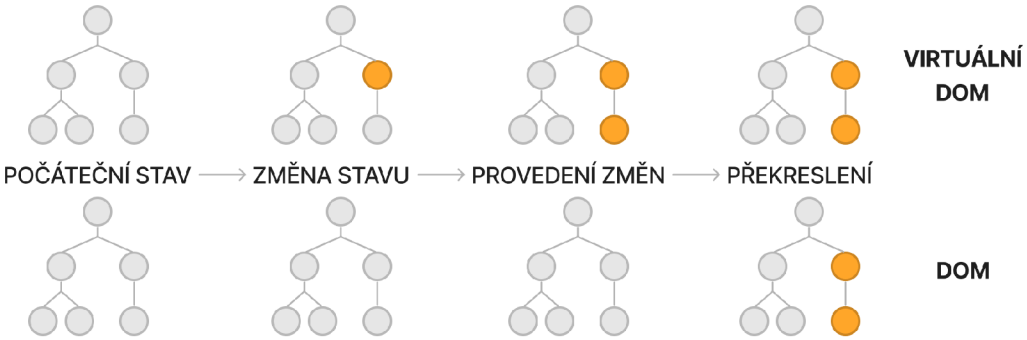
[2] Build proces je proces sestavení aplikace, při kterém dochází např. ke kompilaci kódu, sestavení závislostí a testování.

3.2.1 React

React je JavaScriptová knihovna pro vytváření uživatelských rozhraní. [23] Díky svému konceptu komponent, na kterých jsou založeny React aplikace, byl pro aplikaci skvělou volbou, protože se předpokládalo sdílení UI prvků mezi jednotlivými částmi aplikace.

Výhodou React komponent je kombinování JavaScript logiky a HTML struktury v rámci komponenty. Tento styl zápisu se nazývá JSX. Komponentě je možné předávat různé props[3] a následně je např. zobrazit mezi HTML tagy. Komponenta může být definována jako třída nebo jako funkce a může spravovat vlastní stav.

React využívá virtuální DOM, což je kopie reálného DOMu v paměti. Změny ve virtuálním DOMu aktualizují pouze potřebný uzel DOMu, nikoli celou stromovou strukturu, tedy se neaktualizuje celá stránka (viz. obr. 4). Např. při změně stavu komponenty se aktualizuje pouze tato komponenta se všemi jejími potomky, ale není potřeba aktualizovat rodičovskou komponentu této komponenty.



Obr. 4: Aktualizace DOMu podle virtuálního DOMu.

Ve verzi React 16.8 byly představeny hooky. Hooky jsou callback funkce (návratové funkce), které jsou poskytované Reactem a umožňují funkcionálním komponentám využívat např. stavové chování (komponenta si spravuje vlastní stav) pomocí hooku useState nebo provádění vedlejších efektů na základě změn závislostí pomocí hooku useEffect. Je možné vytvářet i vlastní hooky, které mohou využívat již existující React hooky. Je však nutné dodržovat pravidla a konvence stanovené Reactem, jako je např. používání prefixu „use“ v názvu hooku pro snadnou identifikaci hooku. Hooky mají také pravidla volání v rámci komponent, jako je volání hooku pouze na vrcholu komponenty a ne uvnitř podmínky nebo smyčky. Funkcionální komponenty v kombinaci s hooky představují moderní způsob psaní React komponent, avšak existují situace, kdy je vhodnější použití třídních komponent.

[3] Props je zkrácený výraz pro properties. Jedná se o běžně používaný název v React aplikacích. V podstatě se jedná o parametry předávané funkcím.

3.2.2 Redux, React Redux a Redux Toolkit

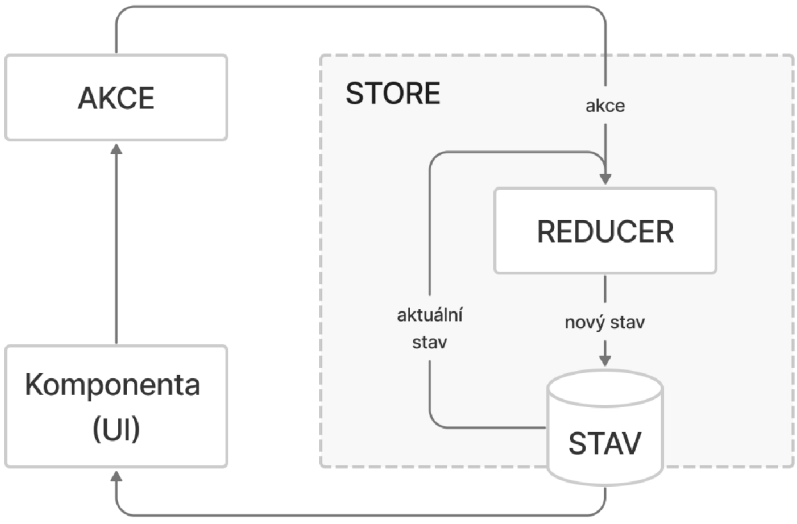
Redux je knihovna, která slouží k spravování globálního stavu aplikace v JavaScriptu a nabízí [24]:

- Předvídatelnost – aplikace má konzistentní chování.
- Centralizovanost – stav aplikace je ukládán centralizovaně.
- Debuggovatelnost – pomocí Redux DevTools je jednoduché sledovat kdy, kde, proč a jak se změnil stav aplikace. Další výhodou je možnost debuggování pomocí „cestování v čase“, což znamená, že je možné aplikaci vrátit do některého z předchozích stavů.
- Flexibilita – dokáže pracovat s jakoukoliv UI vrstvou a má velký ekosystém rozšíření.

Redux má tři základní koncepty:

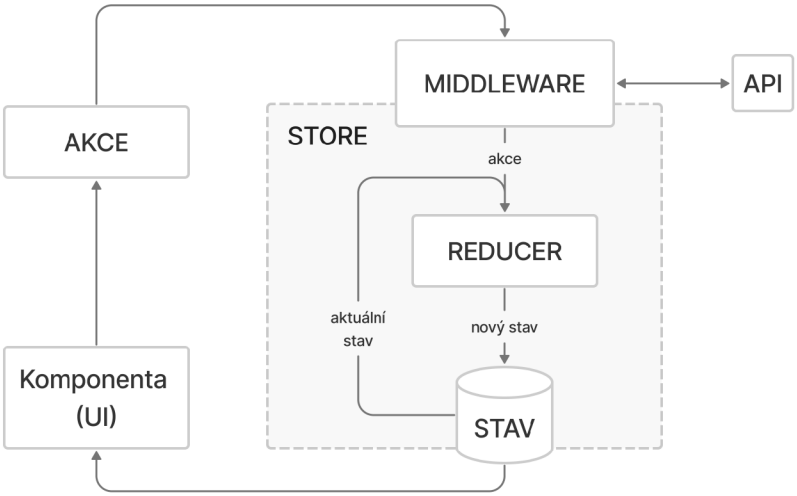
1. Store – JavaScriptový objekt, který reprezentuje stav aplikace. Slouží jako jediný zdroj pravdy, což znamená, že nedochází k nekonzistentnosti dat napříč aplikací.
2. Actions – stav aplikace je pouze ke čtení a nedá se přímo modifikovat. Akce jsou jediný způsob jak změnit stav. Komponenta, která chce změnit stav, musí dispatchnout (odeslat) příslušnou akci.
3. Reducers – funkce, která modifikuje stav na základě akce. Reducer přijme akci a aktuální stav a vrátí aktualizovaný stav.

Pokud tedy chce komponenta změnit Reduxový stav, musí odeslat příslušnou akci. Tato akce je zpracována reducerem, který upraví stav. Komponentám, které odebírají stav se aktualizuje odebíraná hodnota stavu. Proces je vyobrazen na obr. 5.



Obr. 5: Princip Reduxu.

Mezi odesláním akce a zpracováním akce reducerem může být i middleware (prostředník), který zachytí danou akci, provede např. provolání API nebo zaslání analytiky a vyšle novou akci do reduceru. Tento princip je vyobrazen na obr. 6.



Obr. 6: Princip Reduxu s middleware.

Nevýhodou Reduxu je, že vzniká velké množství kódu pro základní konfiguraci, jako např. vytvoření konfigurace úložiště a definice akcí a reducerů. Řešením tohoto problému je knihovna Redux Toolkit. Tato knihovna zavádí nový způsob definice akcí a reducerů pomocí tzv. „slice“. Na základě definice ve slice se automaticky generují akce a reducerové funkce, čímž se snižuje množství psaní potřebného kódu.

3.2.3 React Router

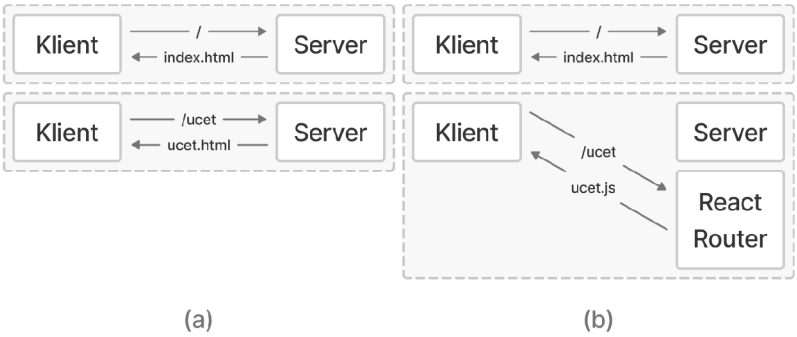
React Router umožňuje routing (směrování) na klientovi. [25] Znamená to, že při kliknutí na link v rámci aplikace se sice aktualizuje URL, ale nedojde k požadavku na server pro získání nového HTML dokumentu (viz. obr. 7a). Místo toho se překreslí React komponenta na komponentu, která je pro tuto URL definovaná (viz. obr. 7b). Díky tomu se získá rychlá odezva a vylepší se UX, protože není potřeba čekat na odpověď serveru.

3.2.4 Leaflet a React Leaflet

Jedná se o knihovny, které slouží k zobrazování mapových podkladů a k uživatelské interakci v zobrazené mapě. Využívají se v kombinaci s mapovými podklady OpenStreetMap.

OpenStreetMap

OpenStreetMap je vytvořeno komunitou napříč celým světem, která přispívá a spravuje data pro mapové podklady. Mapy jsou tak pravidelně aktualizovány. Mapové podklady jsou veřejné a jejich využití není omezeno. [26]



Obr. 7: Stránkování pomocí (a) požadavků na server vs. (b) React Router.

Leaflet

Leaflet je open source JavaScriptová knihovna pro interaktivní mapy a je navrhovaná s ohledem na jednoduchost, výkon a použitelnost. Obsahuje plno funkcionalit a může být rozšířena o další pluginy. [27]

React Leaflet

React Leaflet je knihovna, která nabízí propojení mezi Reactem a Leafletem, tedy nenahrazuje Leaflet, ale vytváří React komponenty zaobalující Leaflet funkcionalitu. [28] Tato knihovna byla vybrána z důvodu jednodušší implementace do Reactu, protože není potřeba vytvářet nové React komponenty.

3.2.5 SheetJS

SheetJS je knihovna, která umožňuje číst, editovat a exportovat např. excel tabulky v rámci webového prohlížeče. [29]

3.2.6 Recharts

Recharts je open source knihovna pro vizualizaci dat. Jedná se o knihovnu poskytující React komponenty, čímž usnadňuje implementaci do projektů postavených na Reactu. Knihovna je postavená na SVG elementech a lehké závislostí na D3.js submodulech. Komponenty lze jednoduše upravovat pomocí předání props. [30]

4 POPIS APLIKACE

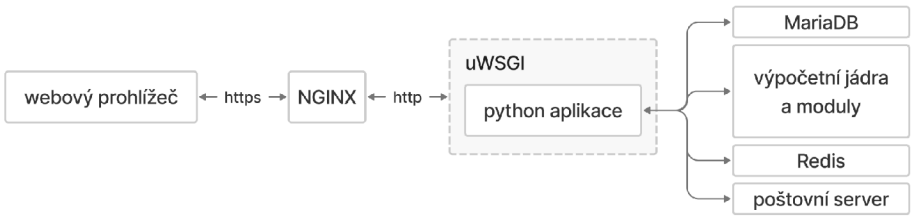
V této kapitole je popsán proces vývoje webové aplikace. Jak již bylo zmíněno na začátku kap. 3, tak aplikace je rozdělena na dvě základní části – frontend a backend. Aplikace procházela postupným vývojem, kdy se k základnímu konceptu aplikace s postupem času přidávaly nové funkcionality. Tyto funkcionality vznikly na základě nových požadavků po testování a konzultacích s konečnými uživateli.

4.1 Základní koncept aplikace

Před zahájením vývoje webové aplikace bylo potřeba stanovit základní koncept aplikace. Tedy jak by měla aplikace fungovat a co by měla obsahovat. Základní návrh vychází z poskytnutého materiálu vedoucím diplomové práce a konzultací. Návrh obsahuje tyto základní body:

- Aplikace poběží za pomoci NGINX a uWSGI.
- Na serveru bude relační databáze s uloženými daty.
- Backend ve frameworku Flask bude vydávat html na klienta a obstarávat ověření a autorizaci uživatelů. Dále bude připojen na výpočetní jádra a další moduly, které bude frontend moci využívat skrze vydefinované API pointy. V neposlední řadě bude backend provádět základní akce s databází (vytvářet, aktualizovat, číst a mazat), které bude poskytovat frontendu pomocí vydefinovaných API pointů.
- Frontend v knihovně React bude zobrazovat data a umožňovat uživatelům manipulaci s daty jako např. zobrazování adresních a sběrných míst v mapě, vytváření a zobrazování výpočtů a mnoho dalších možností.

V průběhu vývoje se aplikace postupně rozšiřovala o další služby jako např. Redis a poštovní server. Koncept aplikace je vyobrazen na obr. 8.



Obr. 8: Zjednodušený koncept aplikace.

Popis jednotlivých částí konceptu:

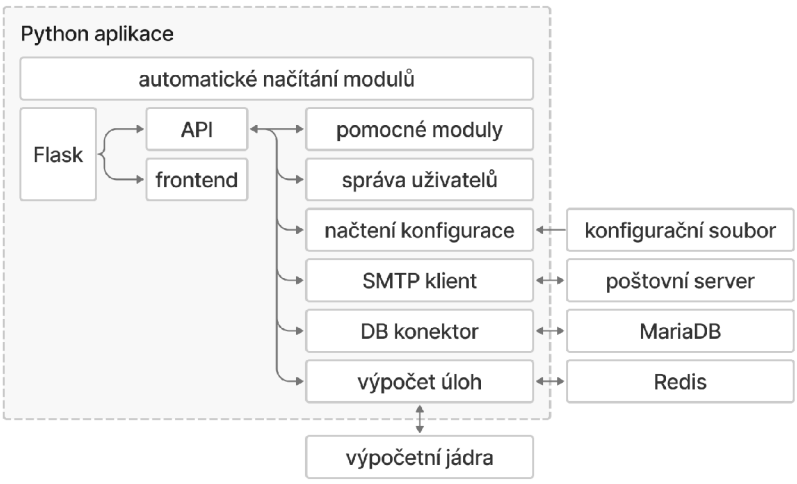
- Webový prohlížeč – uživatelské rozhraní pro zobrazení aplikace (počítač, notebook, větší tablet).

- NGINX – zpřístupnění aplikace uživatelům a šifrování komunikace.
- uWSGI – spouštění více instancí aplikace, pro rozložení zátěže.
- MariaDB – relační databáze pro trvalé ukládání dat.
- Výpočetní jádra a moduly – výpočty pro moduly sběrných míst a svozu odpadu.
- Redis – dočasná data sdílená napříč instancemi aplikace.
- Poštovní server – automatické odesílání e-mailů uživatelům, žádajícím o přístup k aplikaci.

Na obr. 8 a obr. 9 jde vidět, že uživatelské požadavky jsou zpracovávány Python aplikací, tedy beckendem. Tyto požadavky jsou dále rozlišovány na požadavky na aplikaci a požadavky na API pointy. Backend tedy buď odesílá uživateli HTML soubor nebo data ve formátu JSON. Společně s HTML jsou načítány JavaScript skripty a aplikace je vytvořena na klientovi pomocí Reactu.

4.2 Backend aplikace

K vytvoření backendu aplikace se použilo poskytnuté řešení backendu od vedoucího diplomové práce. Z tohoto řešení se převzaly některé moduly, které vytvořily nové jádro backendu, které se následně rozšiřovalo o další moduly. Příklady modulů jsou popsány v kap. 4.2.1.



Obr. 9: Schéma backendu aplikace.

Backend webové aplikace je postavený na Python frameworku Flask a s frontendem komunikuje pomocí definovaných API pointů. Použití backendu v programovacím jazyku Python umožnilo vcelku jednoduchou integraci výpočetních jader, resp. výpočetních funkcí, které jsou tak součástí kódu backendu. Backend dále zajišťuje komunikaci a základní úpravy (vytváření, aktualizování, čtení a mazání) s data-

bází. V průběhu vývoje aplikace se ukázalo, že některá data by bylo lepší si uchovávat dočasně v mezipaměti. Z tohoto důvodu se do projektu zavedl Redis, který navíc nabídl i možnost vytváření front, které se hodily pro řazení a postupné odbavování výpočtů. Dalším rozšířením, které se ukázalo potřebné během vývoje, bylo zavedení e-mailového serveru pro automatické odesílání e-mailů uživatelům, kteří žádají o přístup do aplikace. Došlo tak ke změně z předešlého formátu manuálního vytváření účtů uživatelům administrátory, na vytváření účtů samotnými uživateli a následným požadavkem o schválení přístupu. Řešení ověřování a autorizace uživatelů je prováděno taktéž na backendu a to na úrovni API pointů (více v kap. 4.2.2). Náhled na strukturu backendu je vyobrazen na obr. 9.

4.2.1 Použité moduly na backendu

Použité moduly na backendu by se daly rozdělit do tří základních kategorií:

- moduly zcela převzaté z poskytnutého řešení backendu,
- moduly částečně převzaté z poskytnutého řešení backendu, které byly upraveny pro potřeby aplikace a
- moduly nově vytvořené pro potřeby aplikace.

Níže jsou popsány vybrané moduly.

Konfigurační JSON

```
{
    dbhost: str,            // adresa DB serveru
    dbuser: str,            // DB uživatelské jméno
    dbpassword: str,        // DB uživatelské heslo
    dbdatabase: str,        // název databáze
    session_timeout: int,   // délka platnosti uživatelského připojení v hodinách
    mockup: bool,           // zapnutí mockup api
    smtphost: str,          // adresa SMTP serveru
    smtpport: int,          // SMTP port
    smtpuser: str,          // SMTP uživatelské jméno
    smtpmail: str,          // SMTP uživatelský e-mail
    smtppassword: str       // SMTP uživatelské heslo
}
```

Obr. 10: Struktura konfiguračního souboru.

Pro základní konfiguraci backendu aplikace byl zvolen formát JSON. Konfigurace je uložena v souboru `configuration.json` a načítá se při spuštění aplikace. Soubor není z důvodu bezpečnosti ukládán do systému správy verzí git a pro chod aplikace při lokálním vývoji, nebo na serveru, je potřeba jej vytvořit a manuálně vyplnit. Pokud soubor při spuštění aplikace neexistuje, tak je automaticky vytvořen s prázdným objektem, který je potřeba vyplnit. Na obr. 10 je vyobrazena struktura

konfiguračního souboru s klíči a jejich typy. Protože se může konfigurační soubor lišit při lokálním vývoji a na serveru, je jednoduché např. nastavení připojení databáze, kdy při lokálním vývoji je napojena testovací databáze a na serveru je připojena ostrá verze databáze.

Automatické načítání modulů

Jedná se o převzatý modul, který slouží k automatickému načítání dalších modulů v adresářové složce. Moduly jsou vytvořené v adresářové složce s `__init__.py` souborem, který automaticky načítá moduly s příponou `.py` a `.pyc`. Je-li tedy složka nazvaná např. „utils“ a v ní jsou např. moduly `modul1.py` s funkcí `modul1_function` a `modul2.py` s funkcí `modul2_function`, pak stačí v souboru, kde chceme funkce využít, použít `import utils` a funkce zavolat jako `utils.modul1.modul1_function()` a `utils.modul2.modul2_function()`. Není tedy potřeba importovat každý modul zvlášť.

Konektor do databáze

Jedná se o modul zcela převzatý, který zjednodušuje připojení do databáze. Modul konektoru do databáze je třída obalující funkce knihovny pymysql a zajišťuje tak jednodušší nastavení připojení do databáze. Toto řešení konektoru bylo příhodné a rozhodlo se jej využít, protože splňovalo veškeré aktuální potřeby pro navázání spojení a komunikaci s databází.

Správa uživatelů

Jedná se o modul částečně převzatý a upravený. Součástí modulu je model uživatele, ze kterého se vynechaly některé proměnné, aby byl model zjednodušen pro potřeby aplikace. Jedná se o třídu s proměnnými, které obsahují informace o instanci uživatele. Na obr. 11 je modelový příklad výsledné struktury modelu uživatele, ze kterého jsou vynechány metody třídy.

```
1  class Uzivatel:
2      id: int
3      isadmin: bool
4      login: str
5      name: str
6      rights: set
7      surname: str
8      subject: str
9      token: str
```

Obr. 11: Kód struktury modelu uživatele.

V porovnání s původním modelem se nepracuje s uživatelskými skupinami a pro autorizaci se využívají pouze práva definovaná v `Uzivatel.rights` a hodnota `Uzivatel.isadmin`. Princip autorizace uživatele je popsán v kap. 4.2.2. Proměnná `Uzivatel.token` slouží k ověření uživatelů a odhlášení uživatele, pokud je neaktivní. Pro udržení přihlášení uživatele slouží „keep-alive“ API, která je provolávána automaticky z frontendu ve stanovených intervalech.

Flask JSON odpověď

Jedná se o modul zcela převzatý, který obsahuje funkce zaobalující knihovnu orjson a funkci `make_response` z knihovny flask. Zjednodušuje se tak vytváření odpovědí API pointů při předávání dat v odpovědi, protože stačí použít pouze jednu funkci. Níže je na obr. 12 vyobrazen kód takovéto funkce.

```
1  def json_response(data):
2      json_string = orjson.dumps(data, default=str)
3      return make_response(json_string, {'Content-Type': 'application/json'})
```

Obr. 12: Kód json odpovědi ve Flasku.

Knihovna orjson byla vybrána pro serializaci dat převážně z důvodů vyšší rychlosti než standardní knihovna json v Pythonu, větší bezpečnosti a podpoře více datových typů (např. datetime nebo uuid).

4.2.2 Endpointy

Ve Flask aplikaci se pomocí dekorátorů funkcí přidružují endpointy k funkcím. Příklad přidružení funkce k endpointu je na obr. 13.

```
1  app = Flask(__name__)
2
3  @app.route('/definovany/endpoint', methods=['GET', 'POST'])
4  def nazev_funkce():
5      # Provedení potřebného kódu
6
7      # Návratem funkce je buď HTML šablona nebo data ve formátu JSON
8      return
```

Obr. 13: Kód přidružení endpointu k funkci ve Flasku.

Na obr. 13 je vidět, že definovaný endpoint „/definovany/endpoint“ je zachycen pomocí dekorátoru `@app.route()`. Zpracovávaný endpoint může být definován i dynamicky jako např. „/<path>“, což vede k zachycení čehokoliv za znakem

„/“. Tedy například cesty „/definovany/endpoint“, „/definovany/endpoint-1“ nebo „/definovany-1/endpoint“. Využití API endpointů je popsáno níže v kapitole.

V dekorátoru se dá nastavit i pro jaké metody se má endpoint zachytit. V příkladu je uvedena kombinace „GET“ a „POST“ metod, ale mohou se použít i další metody, jako je „PUT“ a „DELETE“. Lze tedy definovat endpointy pro každou metodu zvlášť, nebo je kombinovat.

V rámci endpointů se provádí ověřování uživatelů. Pokud uživatel není přihlášený, navrátí se HTML s přihlašovací stránkou. V opačném případě se navrátí HTML se stránkou aplikace.

Api endpointy

V aplikaci se odlišují API endpointy pomocí „api“ části v endpointu a jsou řešeny pomocí dynamického endpointu. Mají tedy tvar „/api/<api-point>“. V návratu funkce se volá funkce `api.dispatch(api-point, ...parametry)` z modulu api, která provolá definovaný API point. API pointy jsou definovány ve vlastní souborové struktuře a buď provádí základní akce s databází nebo provolávají výpočetní jádro. Nad těmito definovanými API pointy se provádí autorizace uživatelů. Api pointy mají strukturu, která je naznačena kódem na obr. 14.

```
1  @point({'potrebne_opravneni'})
2  def api-point(...parametry):
3      # Provedení potřebného kódu
4
5      # Navrácení dat ve formátu JSON
6      return
```

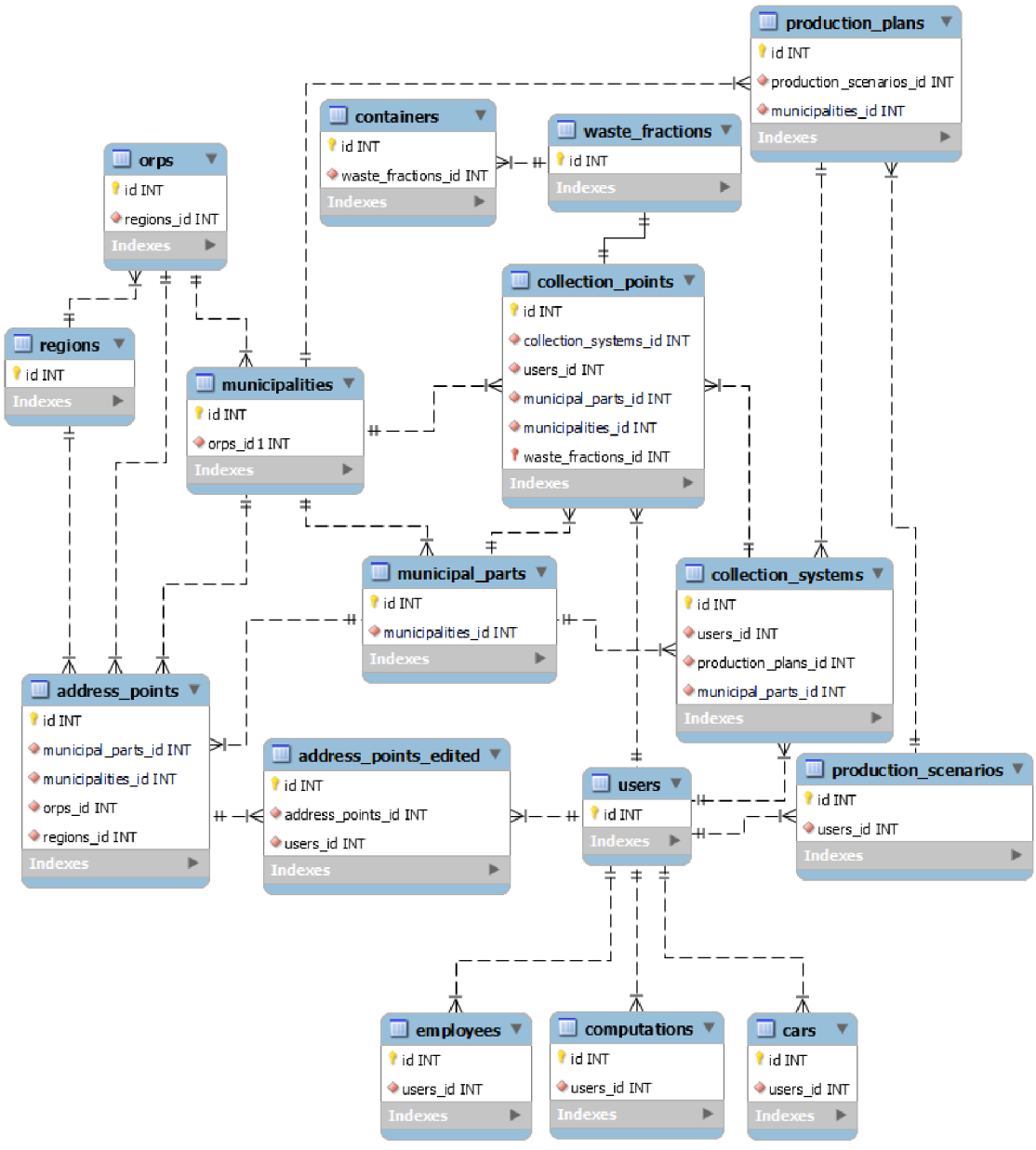
Obr. 14: Kód definice API pointu s oprávněním.

Využívá se zde převzatý dekorátor `@point()`, který vytváří/rozšiřuje slovník API pointů tak, že použije název funkce jako klíč, tedy jako API point, a přidruží k němu funkci a definovaná potřebná oprávnění. Se slovníkem API pointů pak pracuje již zmiňovaná funkce `api.dispatch(api-point, ...parametry)`, která ověřuje, zda aktuální instance uživatele má potřebná oprávnění pro provolání API pointu (uživatel potřebuje mít stejnou nebo vyšší úroveň oprávnění) a pokud ano, tak provolá přidruženou funkci k API pointu ze slovníku API pointů.

Řešení autorizace pomocí specifikace potřebných oprávnění v dekorátoru `@point()` se v průběhu vývoje aplikace prokázalo jako vhodné řešení, když se v aplikaci rozšiřovala oprávnění o demo uživatele, který by měl mít přístup pouze ke specifickým API pointům. Stačilo tedy změnit minimální úroveň oprávnění u těchto API pointů.

4.2.3 Databáze

Při návrhu konceptu aplikace se rozhodlo, že se využije relační databáze, která bude trvale ukládat data. Jako řešení byla vybrána databáze MariaDB. Dalším krokem byl návrh struktury databáze, tedy návrh podoby tabulek a jaké sloupce s jakými typy dat budou obsahovat. Počáteční návrh vzešel na základě poskytnutých dat ve formátu XML a vytvořil se ER diagram, který je ve zjednodušené podobě vyobrazen na obr. 15.



Obr. 15: Zjednodušený ER diagram.

V ER diagramu na obr. 15 lze vidět vazby mezi jednotlivými tabulkami v databázi. Pro přehlednost jsou v tabulkách vyznačeny pouze klíče. Z diagramu lze vyčíst následující:

- Region (regions) obsahuje několik ORP (orps), každé ORP obsahuje několik obcí (municipalities), každá obec obsahuje několik obecních částí (municipal_parts) a každá obecní část obsahuje několik adresních míst (address_points).
- Uživatel (users) může upravovat adresní místa, která se ukládají do tabulky address_points_edited. Adresní místo může mít uživatel upravené několikrát.
- Uživatel může mít definované zaměstnance (employees), auta (cars) a výpočty (computations).
- Uživatel může mít několik produkčních scénářů (production_scenarios). Produkční scénář může mít několik produkčních plánů (production_plans), přičemž produkční plány jsou vázané na obec.
- Uživatel může mít několik sběrných systémů (collection_systems). Každý jeho sběrný systém obsahuje vazbu na jeden produkční plán a několik sběrných míst (collection_points). Sběrná místa jsou vázaná na uživatele, obec, obecní část a frakci sběru (waste_fractions). Frakci sběru může odpovídat několik kapacit kontejnerů (containers).

4.3 Frontend aplikace

Součástí poskytnutého řešení backendu od vedoucího diplomové práce bylo i řešení frontendu, kde Flask zachytává požadavky na endpointy a odesílá HTML na uživatele. Tedy frontend není oddělený od backendu. V poskytnutém řešení byl frontend napsaný převážně pomocí Python knihovny Brython a místy byl využit i JavaScript. Rozhodlo se neinspirovat se řešením pomocí Brythonu, ale jít vlastní cestou a využít pouze JavaScript a to hlavně z důvodu větší znalosti a tak i rychlejšího vývoje. Převzato bylo pouze řešení odesílání HTML z backendu, tedy frontend je řešený pomocí importů skriptů do HTML a neběží samostatně nad Node.js.

Po vzájemné dohodě se frontend postavil na knihovně React, díky čemuž nebylo nutné vlastní řešení AJAX[1]. Pro práci s mapovými podklady byla vybrána knihovna Leaflet a pro jednodušší integraci do Reactu byla vybrána knihovna React Leaflet, která vytváří React komponenty z Leaflet funkcí. K zobrazení dat v grafech byla vybrána knihovna Recharts, která obsahuje React komponenty.

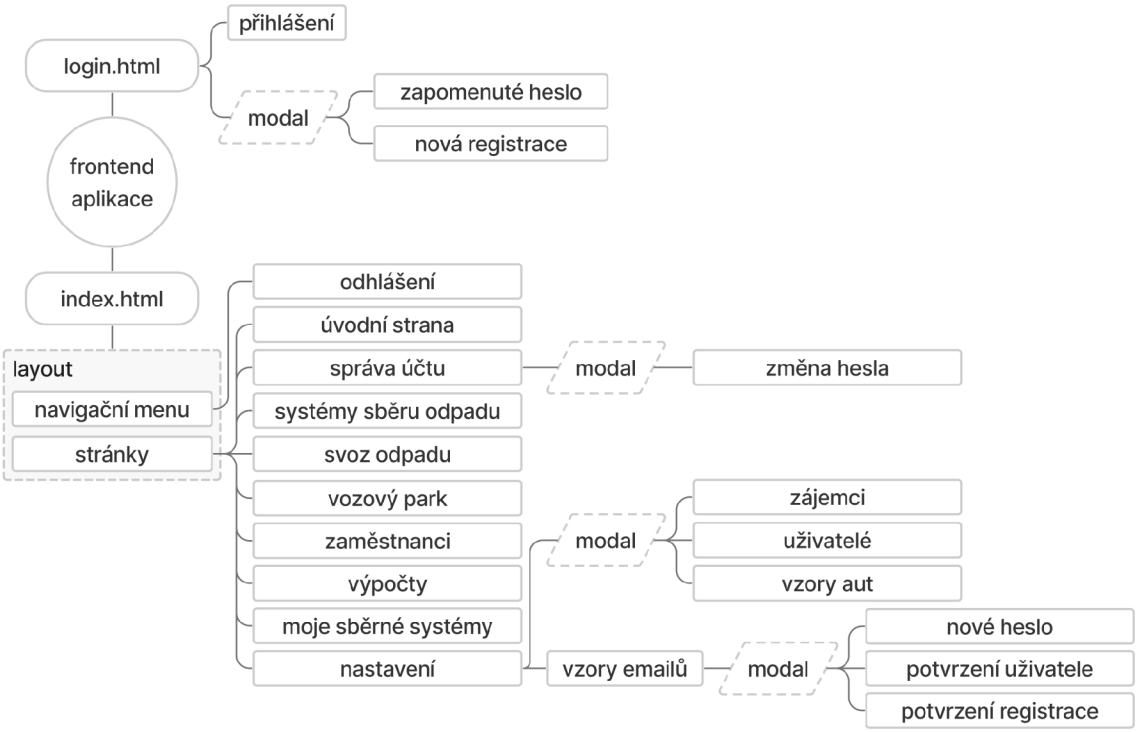
4.3.1 Schéma uživatelského rozhraní

Na obr. 16 je znázorněno stručné schéma uživatelského rozhraní aplikace. V schématu jsou zobrazeny jednotlivé stránky webové aplikace, případně části aplikace zobrazované v modálních oknech. Tato diplomová práce se zabývá primárně vytvořením stránky „systémy sběru odpadu“, která je blíže popsána v kap. 4.3.5 a vytvořením stránky „moje sběrné systémy“, která je blíže popsána v kap. 4.3.6.

Dále je na obrázku vidět, že frontend aplikace je rozdělen do dvou HTML souborů. Pokud uživatel není přihlášený, tak backend odešle HTML šablonu login.html, v rámci které se uživatel může přihlásit do aplikace, obnovit zapomenuté heslo, nebo, pokud se jedná o nového uživatele bez účtu, se registrovat. Po úspěšném přihlášení je z backendu odeslána HTML šablona index.html, která funguje na principu SPA (více v kap. 4.3.2). Obsah obou dvou šablon je vykreslen na klientovi pomocí Reactu.

Na schématu lze dále vidět, že index.html má strukturu obsahující layout, v němž je navigační menu a aktuálně zobrazovaná stránka. Navigační menu se tedy nevytváří v každé stránce zvlášť, ale je součástí layoutu v němž se mění aktuální stránka.

[1] AJAX (Asynchronous JavaScript and XML) není sám o sobě technologií, ale spíše přístupem k využití počtu existujících technologií, včetně HTML nebo XHTML, CSS, JavaScript, DOM, XML, XSLT a XMLHttpRequest objektu. Kombinací těchto technologií v modelu AJAX lze dosáhnout rychlých a přírustkových aktualizací uživatelského rozhraní, aniž by bylo potřeba znovu načítat celou stránku. Díky tomu je aplikace rychlejší a lépe reaguje na akce uživatele. Velmi často se XML nahrazuje použitím JSON. [31]



Obr. 16: Schéma uživatelského rozhraní aplikace.

4.3.2 Routing

Samotná aplikace funguje na principu SPA (single-page application), což znamená, že k routingu dochází v rámci jedné stránky a nejsou odesílány požadavky na server. Tedy, neposílají se požadavky na nový HTML soubor. V prvotních fázích aplikace nebyl zaveden routing v podobě routování přes URL, ale pomocí změn stavu hlavní React komponenty, která na základě tohoto stavu vykreslila dané komponenty. Zjednodušený princip takovéhoto vykreslování React komponent je naznačen kódem na obr. 17.

```
1  switch(state) {
2      case "sber":
3          return <Sber />;
4      case "svoz":
5          return <Svoz />;
6      case "home":
7      default:
8          return <Home />;
9  }
```

Obr. 17: Kód vykreslování stránek pomocí JavaScript switch funkce.

S postupným vývojem aplikace a rozšiřováním o další moduly, např. výsledky výpočtů, se však ukázalo, že by bylo vhodné zavést routing na základě URL se

zachováním SPA konceptu. Jako řešení byl vybrán React Router, který má podobný princip kódu výše, akorát nedochází k využití funkce switch nad hodnotami stavu hlavní komponenty, ale k párování aktuální cesty v URL na komponenty. Takovéto párování je naznačeno v kódu níže (viz. obr. 18), kde se komponentě `Route` předávají props:

- path - cesta, která se chce zachytit a
- component - komponenta, která se má pro danou cestu vykreslit.

```
1  <Switch>
2      <Route path="/sber" component={Sber} />
3      <Route path="/svoz" component={Svoz} />
4      <Route path="/" component={Home} />
5  </Switch>
```

Obr. 18: Kód vykreslování stránek pomocí React Router.

Přechodem na routování pomocí URL se umožnilo odkazovat na moduly pomocí cesty v URL, např. `/sber`. Dalším získaným bodem bylo umožnění uživateli, aby se dostal do jeho výpočtů pomocí URL. Vytváření stránek pro každý výpočet zvlášť by bylo velmi náročné a těžce udržovatelné, a proto se využilo dynamické routování. Při dynamickém routování se vždy vykresluje stejná komponenta, jen se liší zobrazovaná data. Data si komponenta získává dynamicky, dle dynamického parametru v URL. Kód definice dynamické cesty je zobrazen na obr. 19. Komponenta `Vypocet` si získá id výpočtu pomocí hooku useParams a na základě získaného id si natáhne potřebná data z backendu a zobrazí je.

```
1  <Route path="/vypocet/:id" component={Vypocet} />
```

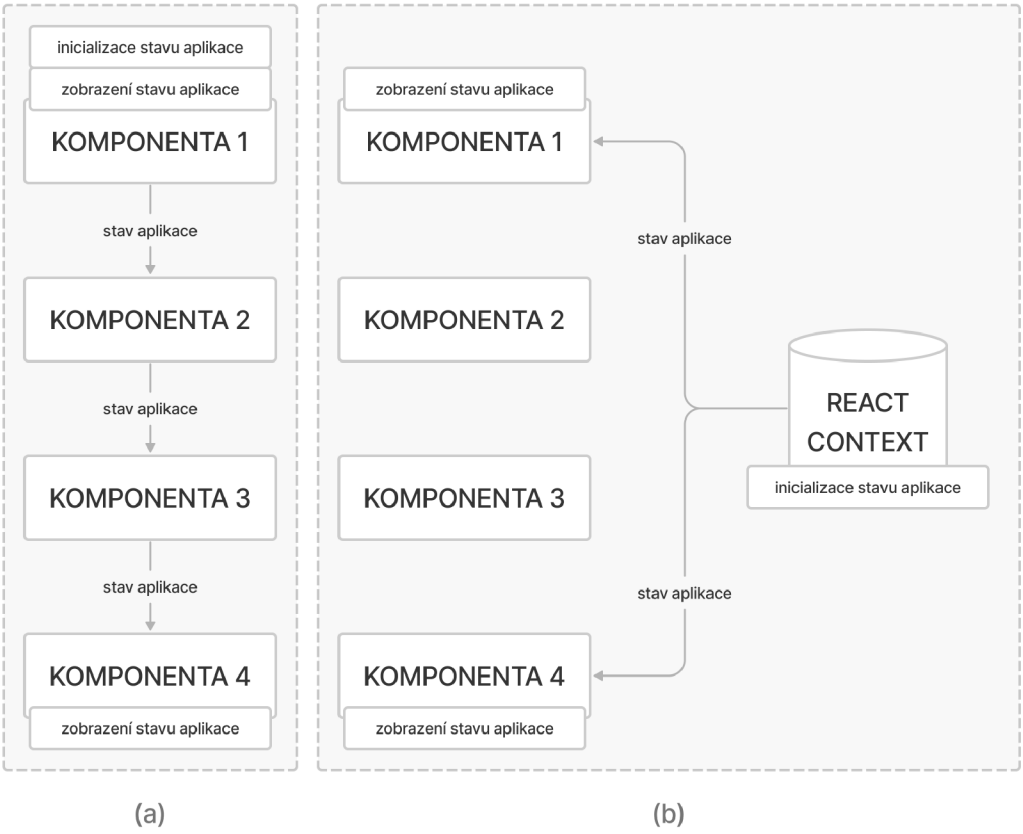
Obr. 19: Kód vykreslování stránek pomocí dynamické cesty.

Další místem, kde se využil nově zavedený routing je navigační menu, které si pomocí hooku useLocation získává aktuální cestu. Cesta se napáruje na název modulu, což umožní zobrazení aktuálně navštíveného modulu (viz. obr. 20).

4.3.3 Správa stavu

V Reactu mohou být komponenty jako funkce nebo třídy. Pokud tyto komponenty spravují vlastní stav, jedná se o stavové funkcionální komponenty, resp. stavové třídní komponenty. Tohoto principu se chtělo využít pro držení informací získaných z backendu. Tedy komponenta odešle požadavek na backend a získá data, která si uloží do

Obr. 20: Zobrazení názvu modulu v záhlaví dle URL (a) /sber a (b) /ucet.



Obr. 21: Naznačení rozdílu mezi (a) „props drilling“ a (b) React Context.

svého stavu. Svůj stav může následně komponenta předávat na své potomky pomocí props, které mohou putovat přes několik generací. Jedná se o proces označovaný „props drilling“. Ze začátku bylo jasné, že může dojít k situacím, kdy je stav využíván komponentou až o několik generací níže. To znamená, že komponentám mezi těmito dvěma komponentami (komponenta se stavem a komponenta zpracovávající stav) bude narůstat počet props, které realně nevyužijí, ale pouze předají o generaci níže. Aby se vyhnulo těmto situacím, využila se funkcionalita Reactu, přesněji se využil React Context. Pomocí React Contextu se může stav definovat globálně a za využití specifických funkcí může jakákoliv komponenta odebírat tento stav, případně provádět změny stavu. Znamená to, že komponenty, které nepotřebují s tímto stavem pracovat, tak jej nemusejí odebírat. Rozdíl mezi použitím „props drilling“

a React Contextu je vyobrazen na obr. 21. Komponenty si tedy spravují vlastní stav pro vlastní potřebu, ale stav sdílený napříč aplikací je obstaráván pomocí React Contextu.

React Context se využil v kombinaci s React hookem useReducer, který umožnil přehlednější správu stavu. Docílilo se toho tak, že se úprava stavu přenesla do speciálních funkcí nazývaných reducery. Samotné komponenty tedy neřeší úpravu stavu pomocí vlastních funkcí, ale odesílají akce, které vyvolají změnu stavu pomocí reduceru. Akce může obsahovat i data, která reducer může zpracovat a promítnout do aktualizovaného stavu.

Kombinace React Contextu a useReduceru se po nějaký čas využívala, ale s předpokládaným růstem aplikace se řešení nejevilo jako nejlepší z dlouhodobého hlediska co se udržovatelnosti a rozšiřovatelnosti týče. Rozhodlo se proto přejít k robustnějšímu řešení v podobě Reduxu, který nabízí komplexnější řešení a hlavně lepší ladění pomocí Redux DevTools rozšíření do prohlížeče. Ovšem nevýhodou Reduxu je zdlouhavý proces nastavení a proto se využila knihovna Redux Toolkit, která tento proces výrazně zjednodušuje. Princip Reduxu je podobný výše zmíněné kombinaci React Contextu a useReduceru, tedy:

1. komponenty odešle akci, která může obsahovat data,
2. akce je zpracována reducerem,
3. reducer aktualizuje stav a
4. komponenta odebírající stav se aktualizuje.

Před použitím Reduxu docházelo k získávání dat z backendu a aktualizaci globálního stavu v komponentách. Z hlediska dlouhodobé udržovatelnosti toto nebylo optimální a proto se postupně přesunula tato logika do Reduxu a to za využití middlewaru. Jako middleware se použila Redux Toolkit funkce `createListenerMiddleware`. Tento middleware ovšem nepracuje imperativně, jak je vyobrazeno na obr. 6, ale reaktivně. Znamená to, že efekt navázaný na akci se provede až jako reakce na aktualizaci Reduxového storu. Komponenty se tak zase o něco zjednodušily, protože nemusely obstarávat získávání, zpracovávání a ukládání dat do stavu, ale jejich primárním úkolem mohlo být zobrazování dat. Nový princip získávání dat je následující:

1. Vytvoří se listenerMiddleware a pomocí funkce `startListening` se začínají poslouchat definované akce, které vyvolají efekty.
2. Komponenta vyšle např. akci `fetchAction`, která ale není registrovaná v reduceru. Data předaná této akci nijak neovlivní stav.
3. Listener čekající na akci `fetchAction` vyvolá svůj efekt.
4. V efektu dojde ke stažení dat z backendu a vyslání akcí za účelem aktualizace stavu.

Na obr. 22 je vyobrazen kód listeneru, který je popsán v principu výše. Z obrázku lze vyčíst, že listener zjednodušuje práci s na sebe navazujícími akcemi. Tedy nejdříve se získají `data1` a vyšle se akce `data1Fetched`, která je zpracována příslušným reducerem, který aktualizuje stav s těmito daty. Následně se získají `data2`, která ve jsou závislá na `data1` a vyšle se akce `data2Fetched`. Tato akce je rovněž zpracována příslušným reducerem, který aktualizuje stav s využitím dat `data2`.

```
1  const listenerMiddleware = createListenerMiddleware();
2
3  listenerMiddleware.startListening({
4      actionCreator: fetchAction,
5      effect: async (action, listenerApi) => {
6          const data1 = await fetch("/api");
7          listenerApi.dispatch(data1Fetched(data1));
8          const data2 = await fetch("/api", {data: data1});
9          listenerApi.dispatch(data2Fetched(data2));
10     }
11 })
```

Obr. 22: Kód vytvoření middleware v Redux Toolkit.

V rámci aplikace slouží Redux i jako mezipaměť na klientovi, kdy v Reduxu jsou uloženy data, která se za běhu aplikace nemění a není tedy nutné opakovat jejich získání. K tomuto v aplikaci slouží inicializační akce, která se vyšle pouze jednou, při inicializaci aplikace na klientovi.

4.3.4 Popis stránek

Rozdělení stránek aplikace je vidět na obr. 16. Design samotných stránek je zaměřen na použití zobrazovacího zařízení o velikosti většího tabletu nebo monitoru a není optimalizovaný pro zobrazování na mobilních telefonech.

Při první návštěvě aplikace, kdy není uživatel přihlášený, je nasměrován na stránku pro přihlášení. Zde má možnost tří dalších akcí:

1. Přihlásit se, pokud má již existující účet.
2. Obnovit zapomenuté heslo.
3. Registrovat se a zažádat tak o přístup do aplikace.

Registrace uživatele funguje na principu schvalovaných žádostí adminy. Uživatel vyplní registrační formulář, po jehož odeslání mu přijde automaticky generovaný e-mail o potvrzení odeslání žádosti.

Rozložení stránek aplikace

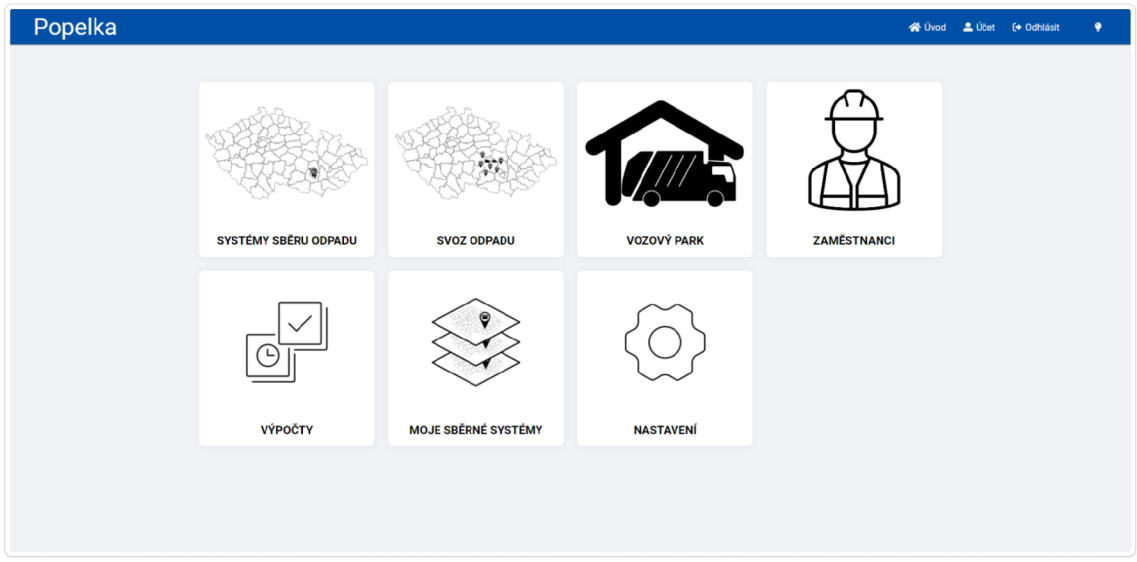
Všechny stránky, kromě přihlašovací stránky, mají sdílený layout (rozložení). Obsahem layoutu je komponenta Navbar, která vytváří navigační menu, a komponenta

stránky, která je vykreslována využitím React Routeru. Tedy komponenta pod Navbarem se mění podle aktuální cesty v URL, ale Navbar stále zůstává a nepřekresluje se.

Úvodní strana

Po úspěšném přihlášení je uživatel přesměrován na úvodní stránku[2] aplikace. Na obr. 23 je vidět rozložení úvodní stránky. Komponenta úvodní stránky vykresluje navigační dlaždice. Slouží tedy jako středobod aplikace, ze kterého se dá navigovat do dalších stránek/modulů. Zobrazené dlaždice jsou pro admin uživatele. Klasický uživatel a demo uživatel nemají přístup do nastavení, a proto u nich tato dlaždice není zobrazována. Moduly do kterých se dá přes dlaždice dostat:

- Systémy sběru odpadu – modul pro plánování rozmístění sběrných míst.
- Svoz odpadu – modul pro plánování svozu odpadu.
- Vozový park – modul pro správu vozového parku
- Zaměstnanci – modul pro správu zaměstnanců.
- Výpočty – modul pro zobrazení výpočtů svozu odpadu.
- Moje sběrné systémy – modul pro zobrazení uživatelem vytvořených sběrných systémů.
- Nastavení – modul pro adminy sloužící ke správě zájemců, uživatelů, vzorů aut a vzorů automaticky odesílaných emailů.

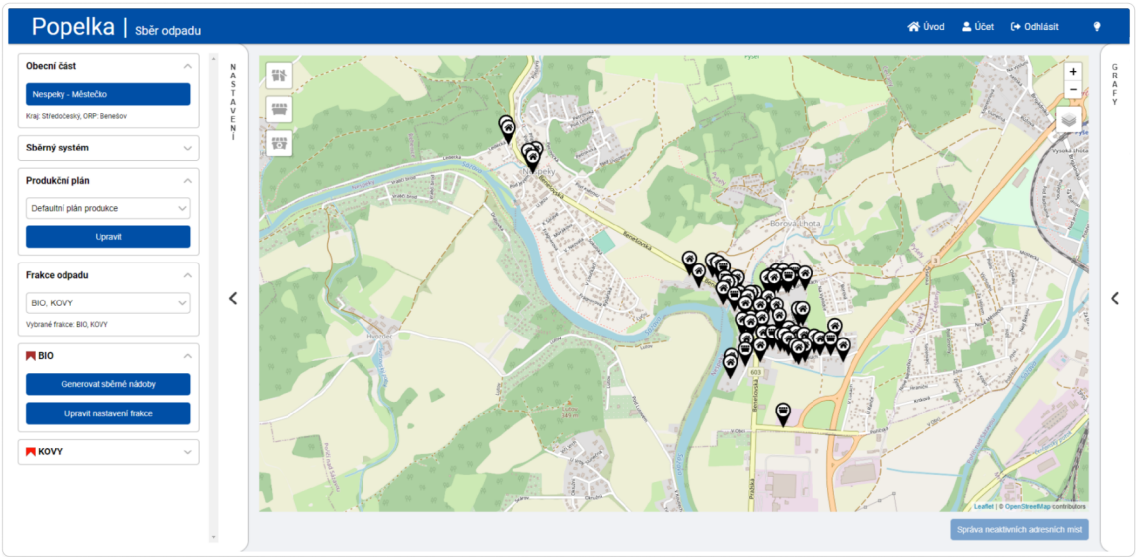


Obr. 23: Úvodní stránka webové aplikace.

[2] Pozn.: Pokud měl uživatel před přihlášením v URL cestu do některého z modulů, je po přihlášení automaticky přesměrován do daného modulu.

4.3.5 Popis stránky pro plánování rozmístění sběrný míst

Tato stránka byla jednou z hlavních náplní diplomové práce. Jedná se o modul, ve kterém si může uživatel vytvořit vlastní sběrnou infrastrukturu. Na obr. 24 je vyobrazena stránka modulu. V levé části je skrývatelný panel nastavení s již vyplněnými parametry v boxech obecní části, sběrného systému aj. Avšak při zadávání parametrů existuje hierarchie, protože některé parametry jsou závislé na výběru jiných parametrů. Např. všechny parametry zadávané v tomto modulu jsou závislé na vybrání obecní části. Takže do doby, než je vybrána obecní část, nelze vybírat další parametry. Největší část stránky zabírá mapa, která slouží k vizuální reprezentaci dat obecní části a sběrného systému. V pravé části je skrývatelný panel s grafy zobrazující statistiky zadání nebo výpočtů.



Obr. 24: Stránka modulu pro rozmístění sběrných míst.

Pozn.: Rozbalený box sběrného systému je na obr. 25. Grafy jsou zobrazeny na obr. 29 a obr. 30.

Na této stránce se také ukázala vhodnost použití Reduxu, aby se reaktivně získávala další potřebná data na základě výběru. Tedy např. po výběru obecní části se z backendu získají data pro sběrné systémy, produkční scénáře a produkční plány, které jsou všechny vázané na obecní část, a jsou tak k dispozici v dalších částech modulu. Další výhodou je, že při změně např. sběrného systému, se změny uloží do databáze, ale není potřeba znovu získávat aktualizovaný daný sběrný systém z databáze pomocí backendu, protože se ty samé úpravy provedou i v Redux stavu. Je tedy ke změnám okamžitý přístup a promítnutí změn není opožděno komunikací mezi klientem a serverem.

Box obecní části

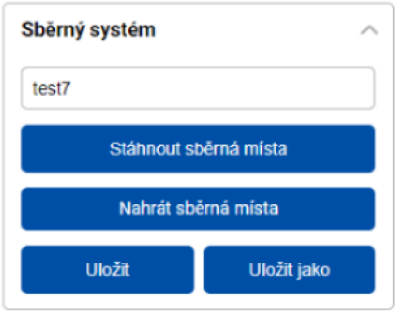
Box obecní části slouží k výběru obecní části ze stromové struktury s vrstvami kraje, orp, obce a obecní části. Je zde také možnost výběru obecní části na základě parametru `municipalityPart` v URL, který obsahuje id existující obecní části. Při opakované práci se stejnou obecní částí je tedy možné zjednodušit proces výběru obecní části právě použitím parametru v URL.

Výběr obecní části ze stromové struktury nebo pomocí parametru v URL vede k vyvolání Redux akce s id obecní části, která je zpracována middlewarem, který provede efekt. V rámci efektu se asynchronně získají data pro obec a obecní část na základě id obecní části. Po získání dat se vyvolá akce pro nastavení obce a obecní části v Reduxu. Dále jsou vyvolány akce pro stažení produkčních plánů a scénářů a stažení sběrných systémů. Obě tyto akce jsou zpracovávány v middlewaru. Na závěr efektu jsou stažena adresní místa obecní části, uložena do Reduxu a zobrazena v mapě.

V efektu, který se vyvolá v reakci na akci pro stažení produkčních plánů a scénářů, se nejprve z Reduxu získá obecní část. Následuje získání produkčních scénářů, které má uživatel vytvořené. Po stažení scénářů dochází ke stažení produkčních plánů, které jsou vybírány na základě id všech scénářů a id obecní části. Následně jsou odfiltrovány produkční scénáře, které se nenachází v získaných produkčních plánech a produkční plány a scénáře jsou uloženy do Reduxu.

Na základě akce pro stažení sběrných systému je vyvolán efekt, ve kterém se získá id obecní části z Reduxu a následně jsou staženy sběrné systémy pro dané id obecní části. Po stažení jsou sběrné systémy uloženy do Reduxu.

Box sběrného systému



Obr. 25: Box nastavení sběrného systému.

V boxu sběrného systému si může uživatel vybrat z již existujících sběrných systémů, které si vytvořil a jsou uložené v Reduxu na základě výběru obecní části. Uživatel může zadat i nový název sběrného sytému, pokud chce vytvořit nový sběrný systém. Po zadání názvu, ať už existujícího nebo nového, má uživatel nabídku dalších akcí, která je vidět na obr. 25. K zadání názvu sběrného sytému je

možné využít parametr v URL `collectionSystem`, který obsahuje název sběrného systému. V kombinaci s parametrem v URL pro výběr obecní části vytváří jednoduchý a rychlý způsob častého výběru stejné obecní části a stejného sběrného systému. Pokud sběrný systém neobsahuje sběrná místa, je uživatel notifikován pomocí toast notifikace (viz. služba zobrazování toastů v kap. 4.3.7).

První nabízenou akcí po výběru sběrného systému je stažení vygenerované excel šablony, která je vyplněná všemi sběrnými místy sběrného systému. V této šabloně může uživatel přidávat, upravovat nebo mazat sběrná místa. Další možnou akcí je nahrání šablony. Nahrávaný soubor musí odpovídat struktuře šablony nabízené ke stažení. Po nahrání excelu se správnou strukturou je provedena kontrola nad buňkami excelu, které jsou povinné při vytváření sběrných míst. Pokud soubor projde kontrolou, může si jej uživatel uložit a tím uložit sběrná místa v databázi, kde nad novými sběrnými místy se provede `INSERT`, nad existujícími sběrnými místy se provede `UPDATE` a nad smazanými sběrnými místy se provede `DELETE`. Při kliku na uložit se také nastaví aktuálně vybraný produkční plán jako výchozí produkční plán sběrného systému. Pokud by chtěl uživatel sběrný systém uložit pod jiným názvem, je zde možnost uložit jako.

```
1  export const setCollectionSystemEffect = async (action, listenerApi) => {
2      let collectionSystems = collectionSystemsSelector(listenerApi.getState());
3      ...
4
5      if (!collectionSystems) {   // collectionSystems je null nebo []
6          const [, currentState] = await listenerApi.take((action, currentState) =>
7              collectionSystemsSelector(currentState)
8          );
9          collectionSystems = collectionSystemsSelector(currentState);
10     }
11     ...
12 }
```

Obr. 26: Kód čekání v efektu na nastavení hodnoty v Reduxu.

Výběr sběrného systému ručním zadáním nebo automaticky pomocí parametru v URL vede k vyvolání Redux akce s názvem sběrného sytému. Tato akce je zpracovávaná middlewarem a vyvolává příslušný efekt. V efektu jsou z Reduxu získány potřebná data: sběrné systémy, obecní část, produkční plány a produkční scénáře. Jelikož je možné, že je tento efekt vyvolán dříve, než jsou potřebná data v Reduxu uložena (pravidelný jev při načítání sběrného sytému pomocí parametru v URL), je zde využita funkcionalita nabízená v rámci efektů `take`, která čeká, než dojde ke změně hodnoty v Reduxu (viz. obr. 26). Každá z hodnot je takto kontrolovaná zvlášť, čímž se odstranilo riziko zaseknutí čekání. Funkci `take` je možné také nastavit timeout parametr, po jehož uplynutí přestane funkce čekat a nedojde

k zaseknutí. V praxi se ukázalo lepší rozdělit kontrolu a tím zajistit, že je každá hodnota k dispozici, což by nemuselo být splněno při použití timeoutu.

Následně je ve sběrných systémech získaných z Reduxu nalezen sběrný systém se stejným názvem, jako je název předávaný v akci. Pokud zde není název nalezen, jedná se o nový sběrný systém, kterému jsou nastaveny základní hodnoty a je mu nastaven výchozí produkční plán. Poté je sběrný systém uložen do Reduxu společně s produkčním plánem, který je nastaven jako výchozí u sběrného sytému, a produkčním scénářem, který odpovídá výchozímu produkčnímu plánu. Pokud se jedná již o existující sběrný systém, jsou získána všechna příslušná sběrná místa z databáze a uložena do Reduxu. Pokud u sběrného systému neexistují sběrná místa, je vyvoláno zobrazení notifikace o této skutečnosti.

Box produkčního plánu

Výchozí produkční plán je nastavený na základě vybraného sběrného systému. Pokud se jedná o nový sběrný systém, tak je jako výchozí produkční plán automaticky nastaven „Defaultní produkční plán“ pro zvolenou obecní část. Uživatel může vytvářet a upravovat produkční plány. Uložením sběrného systému se naváže aktuálně vybraný produkční plán na sběrný systém a stává se z něj výchozí produkční plán sběrného systému.

Při ukládání produkčního plánu je vyvolána Redux akce, pro kterou se efekt pokusí o uložení produkčního plánu v databázi. Řešení volání API na uložení produkčního plánu je v efektu z důvodu, že v závislosti na úspěšné či neúspěšné aktualizaci je vyvolána akce pro zobrazení notifikace, která informuje uživatele o stavu aktualizace. Aktualizovaný produkční plán není potřeba opětovně získávat z databáze, protože při úspěšné aktualizaci se provede změna i v Reduxu.

Při vytváření nového produkčního plánu je opět vyvolána Redux akce, na kterou je navázaný efekt, který provolá API na vytvoření nového produkčního plánu. O výsledku vytvoření, tedy jestli se podařilo nebo nepodařilo vytvořit nový produkční plán, je uživatel notifikován pomocí toast notifikace. Pokud došlo k úspěšnému vytvoření, tak z API přijde v odpovědi i nově vytvořený produkční plán, který se nastaví do Reduxu jako aktuální a přidá se do seznamu produkčních plánů.

Box frakcí odpadu

V tomto boxu si uživatel vybírá frakce, se kterými chce dále pracovat. Na základě vybraných frakcí se zobrazí box pro každou z frakcí. Pro zjednodušení opakovaného výběru je zde možnost použití parametru v URL `fraction`, který obsahuje název frakce, která má být vybrána. Tento parametr se může v URL opakovat, protože se získávají všechny jeho výskyty v URL, aby bylo možné vybrat několik frakcí najednou.

Box frakce odpadu

Jedná se o box, ve kterém se pracuje na úrovni frakce odpadu. Uživatel má zde dvě možnosti akce. První možností akce je nechat si vygenerovat sběrná místa. Další možnou akcí je upravení nastavení frakce.

Pro generování sběrných míst se využívá výpočetní jádro na backendu. Aktuálně dochází ke generování sběrných míst na pozici existujícího adresního místa. Je možné vybrat mezi jedním ze čtyř typů generování:

- Centralizovaně – způsob generování, kdy je vytvořeno jedno sběrné místo v rámci obecní části, které je umístěno na adresní místo přibližně ve středu všech adresních míst obecní části. Uživatel si zde pro výpočet definuje frekvenci svozu, preferované dny svozu a způsob výsypu.
- Distribuovaně – způsob generování, kdy je vytvořeno několik sběrných míst na existujících adresních místech tak, aby pokryly produkci adresních míst v obecní části. Uživatel si zde pro výpočet definuje frekvenci svozu, preferované dny svozu a způsob výsypu.
- Door-to-door – způsob generování, kdy na každém adresním místě je vytvořeno jedno sběrné místo. Uživatel si zde pro výpočet definuje frekvenci svozu a preferované dny svozu.
- Vlastní – způsob generování, který má uživatel více pod kontrolou, protože si může zadat více parametrů pro výpočet. Uživatel si zde pro výpočet definuje frekvenci svozu, preferované dny svozu, způsob výsypu, typ sběrného místa, počet kontejnerů, velikost kontejnerů, minimální počet obyvatel na adresním místě a maximální počet obyvatel na adresním místě.

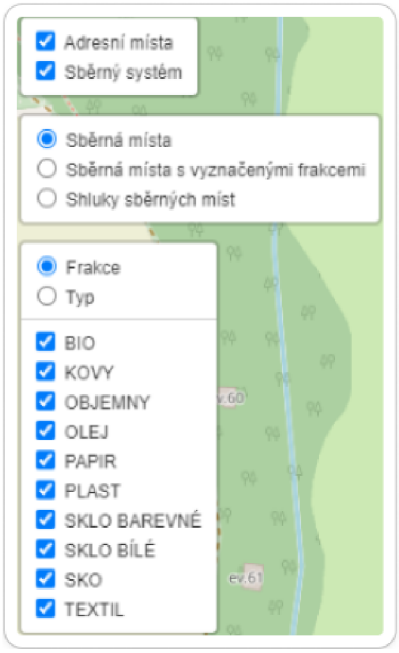
Součástí požadavku na API výpočetního jádra jsou i další parametry, které se získávají z Reduxu. Jedná se o název sběrného systému, id frakce, název frakce, id obecní části a roční produkce frakce získaná z produkčního plánu.

Mapa

Mapa (viz. obr. 24) slouží k zobrazování adresních a sběrných míst, která se zobrazují automaticky na základě listů uložených v Reduxu. Adresní místa jsou získána na základě výběru obecní části a sběrná místa jsou získána na základě výběru sběrného systému. V pravé horní části mapy si může uživatel nastavit přiblížení mapy a změnit mapové podklady. V levé horní části se nacházejí filtry (viz. obr. 27) zobrazení míst v mapě:

- Filtr pro zobrazení adresních a sběrných míst.
- Filtr pro zobrazení sběrných míst, sběrných míst s vyznačenými frakcemi odpadu nebo shluků sběrných míst. Možnost filtrování podle shluků je k dispozici pouze pokud sběrný systém obsahuje shluky, které musejí být vygenerované.

- Filtr pro zobrazení sběrných míst podle frakce odpadu nebo typu sběrného místa. Možnosti jsou nabízeny dynamicky na základě sběrných míst, tedy např. možnost frakce bio je nabízena pouze pokud existuje alespoň jedno sběrné místo s touto frakcí.

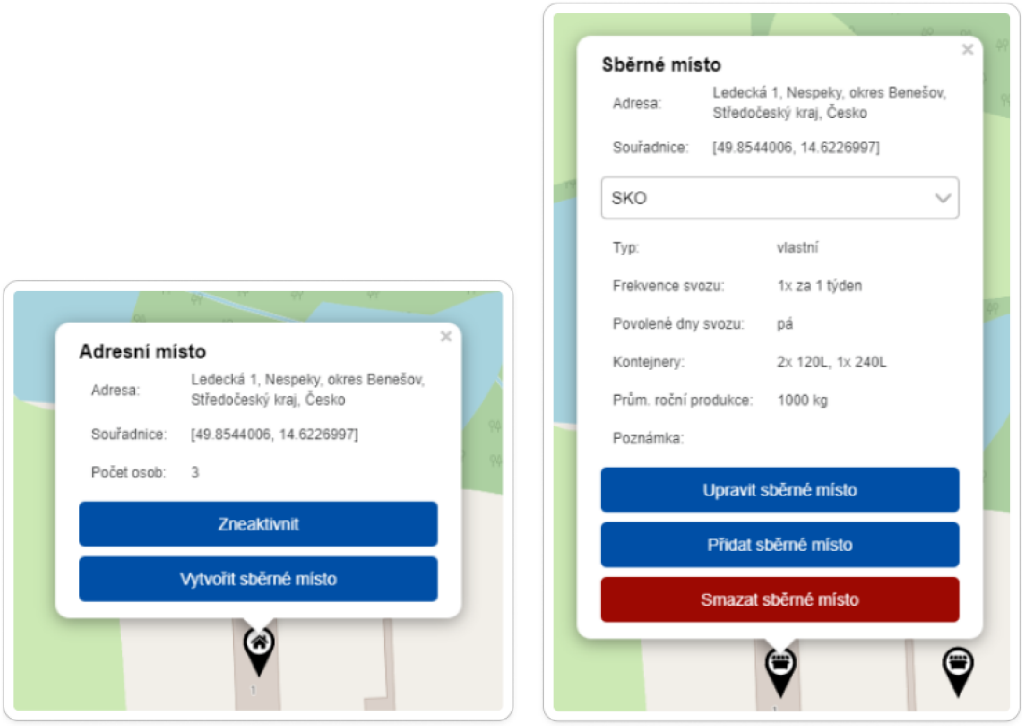


Obr. 27: Filtry pro zobrazování v mapě.

Adresní a sběrná místa v mapě

Adresní i sběrná místa jsou v mapě interaktivní a po kliku se otevře popup, který je vyobrazen na obr. 28. V popupu je u adresních míst možnost zneaktivnění adresního místa. Tedy adresní místo není zahrnuto do výpočtů jako např. generování sběrných míst v boxu frakce odpadu. Dále je zde možnost vytvoření sběrného místa z adresního místa. K vytvoření sběrného místa je potřeba vyplnit formulář doplňující sdílená data mezi adresním místem a sběrným místem. V popupu sběrného místa má uživatel možnost editace sběrného místa, vytvoření dalšího sběrného místa nebo smazání aktuálního sběrného místa. K vytvoření a editaci slouží stejný formulář jako u adresního místa, akorát sdílená data jsou tentokrát převzata z existujícího sběrného místa.

Při zneaktivnění adresního místa dochází k úpravě parametru adresního místa a zapsání do seznamu neaktivních adresních míst v Reduxu. Adresní místo následně může být zase aktivováno, čímž dojde k úpravě parametru adresního místa a vymazání ze seznamu neaktivních adresních míst. List neaktivních adresních míst je vytvářen z důvodu jednodušší manipulace při správě neaktivních adresních míst.



(a) (b)

Obr. 28: Popup (a) adresního místa a (b) sběrného místa.

Pro vytváření sběrného místa z adresního místa, i při úpravě nebo vytváření sběrného místa z existujícího sběrného místa, dochází k vyvolání Redux akce s efektem. Při neúspěšném uložení sběrného místa je vyslána notifikace o neúspěchu. Při úspěšném uložení sběrného místa, je uživatel informován o úspěchu a jsou opět stažena sběrná místa z backendu a uložena do Reduxu.

Smazání sběrného místa vyvolá Redux akci, na kterou je navázaný efekt, který nejprve smaže sběrné místo v databázi a při úspěchu se zmaže sběrné místo i z Reduxu. O úspěchu nebo neúspěchu smazání je uživatel notifikován. Je zde opět využita funkce `take`, která čeká na dokončení odstranění sběrného místa z Reduxu a následně, pokud je seznam sběrných míst prázdný, informuje uživatele o této skutečnosti.

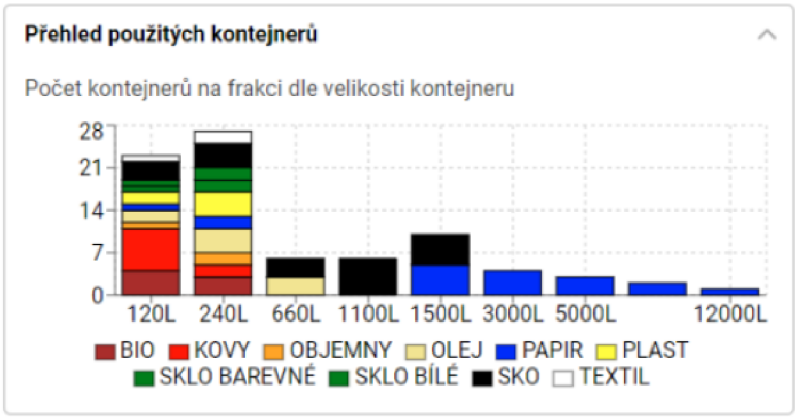
Správa neaktivních adresních míst

Jedná se o modální okno, ve kterém jsou zobrazeny všechny adresní místa obecní části, která si uživatel v mapě zneaktivnil. Zneaktivněná adresní místa jsou uložena seznamu neaktivních adresních míst v Reduxu. Běžný uživatel si zde může tato adresní místa uložit do databáze, takže si adresní místa při dalších načteních drží informaci o svém zneaktivnění. Administrátor má zde navíc možnost „smazat“ adresní místo, což znamená, že u adresního místa v databázi změní parametr a adresní místo je vynecháno z dotazu na adresní místa obecní části pro všechny uživatele.

Aby se smazání adminem promítlo i v mapě bez potřeby stahování dat z backendu, je z Reduxu odstraněno vybrané adresní místo.

Graf použitých kontejnerů

Jedná se o graf, který zobrazuje přehled použitých kapacit kontejnerů u každé z frakcí (viz. obr. 29). Každá frakce je v grafu barevně odlišena pro zvýšení přehlednosti. V grafu lze také vyznačit pouze jednu z frakcí a to pomocí kliku na požadovanou frakci v legendě grafu. Takto se zvýrazní zvolená frakce a nad blokem frakce se zobrazí počet kontejnerů této frakce. Počet kontejnerů dané kapacity pro každou z frakcí lze zobrazit i najetím myši nad požadovaný sloupec.



Obr. 29: Graf přehledu použitých kontejnerů.

Graf docházkové vzdálenosti

Zobrazovaný graf na obr. 30 je pro každou z frakcí zvlášť. Pro zobrazení grafu je nejprve nutné vybrat frakci v levém panelu nastavení a následně kliknout na tlačítko pro načtení docházkové vzdálenosti. Klik vyvolá přepočet docházkové vzdálenosti u dané frakce a zobrazení grafu. V grafu je zobrazena vzdušná docházková vzdálenost na počet obyvatel v jednom adresním místě. Pro referenci je v grafu také znázorněna průměrná docházková vzdálenost dané frakce.

4.3.6 Popis stránky pro výpis a správu sběrných systémů

Tato stránka byla další z hlavních náplní diplomové práce. Jedná se o modul, který slouží k zobrazení uživatelem vytvořených sběrných systémů. Uživatel má zde tabulku, ve které je stručný přehled všech sběrných systémů napříč všemi obecními částmi. V tabulce jsou informace o poslední aktualizaci sběrného systému, název obce, název obecní části, název sběrného systému, nádoby a shluky. Ve sloupci nádoby si může uživatel rozkliknout buňku a zobrazit si počty jednotlivých nádob u každé frakce. Ve sloupci shluky si může uživatel rovněž rozkliknout buňku a zobrazit si počty shluků u každé frakce. Každý záznam v tabulce má dále možnost

Obr. 30: Graf docházkových vzdáleností frakce.

smazání, čímž si uživatel smaže daný sběrný systém i se všemi sběrnými místy, která jsou na tento sběrný systém navázána, a také možnost zobrazení sběrného systému v mapě, což vede k přesměrování do modulu sběru a automatickému zobrazení sběrného sytému.

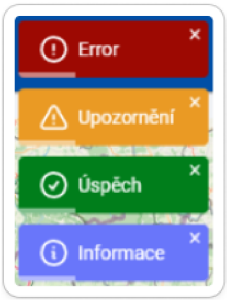
V rámci hlavičky se dají navíc data seřadit v sloupcích poslední aktualizace, obce, obecní části a sběrného systému vzestupně nebo sestupně. Filtr je na principu FIFO (First In, First Out) což znamená, že priorita filtru závisí na jeho pořadí zadání. Tedy dá se řadit podle data poslední aktualizace a následně podle názvu sběrného systému, nebo také naopak, na základě pořadí zadání daného filtru řazení. Svou prioritu si filtr zachovává i při přepnutí z řazení vzestupného na řazení sestupné.

4.3.7 React komponenty

V této kapitole budou popsány komponenty vytvořené v rámci této diplomové práce, nikoliv komponenty vytvořené v [5].

Služba zobrazování „toastů“

Na obr. 31 jsou vyobrazeny všechny podoby toastu. Jak je vidět, tak toast je v podstatě popup s krátkou zprávou, která má určitý charakter zprávy (error, upozornění, úspěch, informace). Toast má nastavenou dobu zobrazení, tedy po krátké době sám zmizí, pokud není uživatelem odstraněn pomocí křížku. Služba je vytvořená pomocí Reduxu. Tento přístup umožňuje zobrazování toastu na základě akce. To znamená, že akci pro zobrazení toastu může vyvolat komponenta např. po kliku na tlačítko nebo tato akce může být vyvolána v reakci na stažená data v efektu listeneru. Její použití je tedy jednoduché, stačí vyslat akci požadovaného charakteru se zprávou, která se má zobrazit. Zpráva nemusí být pouze textová, může být poslána i React komponenta, tedy v rámci toustu se může zobrazit např. i odkaz.



Obr. 31: Zobrazení toastů v aplikaci.

Pro zobrazování toastů je potřeba vložit ToastServiceProvider komponentu do indexové komponenty. Komponenta ToastServiceProvider odebírá z Reduxového stavu list toastů a na základě něj zobrazuje toasty. Toast je samostatná komponenta, v níž se po zobrazení spustí timer, po jehož uplynutí je toast odstraněn z Reduxového stavu.

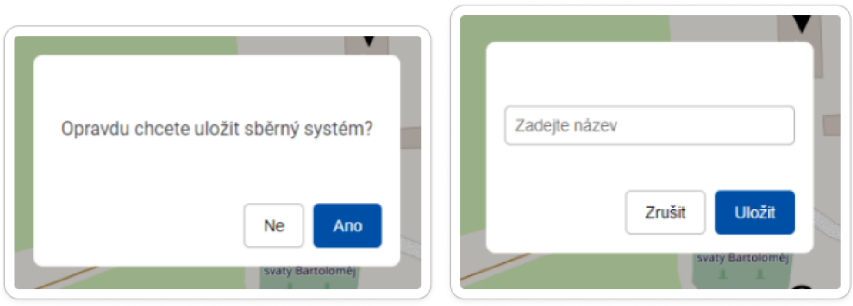
Dialogy pomocí React hooků

Pro zobrazování dialogů (viz. obr. 32) jsou vytvořeny vlastní React hooky, které zaobalují a rozšiřují komponentu Dialog. Návratem hooku je funkce pro otevření dialogu a komponenta dialogu. Komponenty využívající dialog tak nemusí samy řešit vlastní stav pro zobrazování dialogu a pouze využijí poskytnutou funkci. Zde jsou uvedeny vytvořené hooky:

- useConfirmationDialog – komponenta rozšiřující Dialog o text, která slouží k potvrzení provedení akce. Komponentě se předává text, který se má zobrazit v dialogu a callback funkce, která se má zavolat při potvrzení.
- useInputDialog – komponenta rozšiřující Dialog o textové pole a slouží pro zadávání textu. Tento hook využívá useConfirmationDialog hook pro potvrzení zadaného textu. Zajišťuje také ověření, že pole není prázdné. Pro notifikaci prázdného pole se využívá toast notifikace. Komponentě se předává text a callback funkce, které se předávají komponentě generované z hooku useConfirmationDialog, a nepovinný list textů, nad kterým se provádí kontrola zadaného textu. Pokud je text obsažen v listu, opět se odešle akce pro zobrazení toastu.

Modální okna pomocí React hooku

Jedná se o hook, který zjednodušuje používání modálních oken. Návratem hooku je funkce pro zobrazení modálního okna a komponenta modálního okna. Tedy komponenta využívající hook si nemusí spravovat vlastní stav pro zobrazování modálního okna.



Obr. 32: Dialog (a) potvrzovací a (b) s inputem.

Navigační menu

Pro zobrazení navigačního menu (viz. obr. 33) slouží komponenta Navbar. Komponenta v levé části zobrazuje logo, přes které se dá navigovat na úvodní stránku. Dále se vedle loga zobrazuje aktuálně navštívený modul, čehož je docíleno pomocí získání cesty z URL a napárování na příslušný název modulu. V pravé části komponenty se nachází navigační tlačítko na úvodní stránku, navigační tlačítko na stránku správy účtu, kde si může uživatel např. změnit heslo, tlačítko pro odhlášení a tlačítko pro přepínání barevného tématu aplikace mezi světlým a tmavým.

Obr. 33: Navigační menu.

Zavíratelné boční menu

Komponenta Expander slouží k zaobalení obsahu na levé nebo pravé straně stránky a umožňuje skrývání obsahu. Je využívána v modulu sběru i svozu. Má dvě možnosti rozbalování. První možností je roztáhnutí v prostoru, tedy obsah vedle komponenty se posune nebo zmenší. Druhou možností je překryv přes obsah vedle komponenty, tedy obsah vedle komponenty nemění svou pozici nebo tvar. Při otevření/zavření komponenta odesílá event o změně stavu, který je zpracováván v komponentě mapViewController.

Box v nastavení modulu sběru

Na obr. 24 jsou v levé části vyobrazeny boxy i ve svých dvou stavech – otevřený a zavřený. Komponenta zobrazuje nastavený titulek a obsah. Je také možné komponentě předat prop barvy k zobrazení barevného štítku vedle titulku, což je využíváno u boxů jednotlivých frakcí.

Mapa

Komponenta mapy je postavena na knihovně React Leaflet a využívá její komponenty. V rámci této komponenty je zobrazována mapa, filtry mapy, adresní a sběrná místa v mapě.

Filtry v mapě

Jedná se o komponenty rozšiřující komponenty poskytované knihovnou React Leaflet a slouží k filtrování adresních a sběrných míst v mapě. Takto se vytvořily filtry přímo na míru potřebám. Filtry se dají i jednoduše rozšířit.

Překreslování mapy

Komponenta mapViewController zajišťuje překreslení v mapě na základě změny adresních nebo sběrných míst. Tedy zpracuje všechny místa a nastaví zobrazení mapy tak, aby byla všechna místa zobrazena. Komponenta má také event listener na event vyslaný z komponenty Expander a na základě něj vyvolá invalidaci velikosti mapy. Je to z toho důvodu, že když se zavře Expander a změní se velikost mapy, tak aby v mapě nezůstávala prázdná šedá místa o velikosti předtím otevřeného Expanderu.

Marker

Marker (značka v mapě) má dvě podoby – adresní místo a sběrné místo. Jedná se o dvě komponenty z důvody zobrazování rozlišných popupů (viz. obr. 28). Aby mohli mít markery vlastní ikonku, bylo potřeba vytvořit komponentu MarkerIcon, která využívá knihovnu Leaflet pro zaobalení SVG ikonky do divu, který dokáže komponenta Marker z knihovny React Leaflet zobrazit místo výchozí ikonky. Kromě rozdílných popupů je i rozdíl v zobrazovaných markerech, kde sběrné místo dokáže vedle ikonky navíc zobrazit jednotlivé frakce (viz. obr. 34), které danému sběrnému místu přísluší.

Obr. 34: Markery sběrných míst s vyznačenými frakcemi.

Atomické komponenty

Jsou to komponenty, které se už nedají dělit na menší celky. Jedná se o následující komponenty:

- Button – komponenta tlačítka, kterému se mění vzhled a funkcionalita na základě předaných props. Tímto způsobem jsou tlačítka napříč modulem jednotná.

- Checkbox a Radio – komponenty, které rozšiřují klasický html input typu checkbox a radio o vlastní vzhled.
- Multiselect – komponenta, která umožňuje výběr více hodnot z nabízených možností. Komponenta využívá komponentu Checkbox.
- NumberInput – komponenta upravuje klasický input typu number.
- Select – komponenta, která umožňuje výběr jedné hodnoty z nabízených možností. Komponenta využívá komponentu Radio.
- TextInput – jednoduché textové pole.
- TextInputWithSelect – komponenta umožňuje zadávání textu a zároveň možnost výběru z nabízených možností.

4.3.8 Pomocné funkce

Při vývoji vzniklo několik pomocných funkcí, jednak aby se obsah React komponent udržel co nejmenší, nebo aby se zjednodušila práce.

Vlastní hook pro odesílání akce

Funkcionální komponenty mohou využívat React hooky. React Redux poskytuje hook na odesílání akcí useDispatch, který ovšem není memoizovaný[3], ale dá se memoizovat pomocí React hooku useCallback. Aby se každá dispatchnutá akce nemusela zaobalovat do useCallbacku, tak se vytvořil vlastní hook. Kód hooku je vyobrazen na obr. 35.

```
1  export const useAction = action => {
2      const dispatch = useDispatch();
3
4      return useCallback(payload => dispatch(action(payload)), [dispatch, action]);
5  };
```

Obr. 35: Kód vlastního React hooku useAction.

Funkce generování excelu

Jedná se o pomocnou funkci, která vygeneruje excel soubor vyplněný na základě předaných parametrů pomocí knihovny SheetJS. Tato funkce je využívána v boxu sběrného systému. Vygenerovaný excel slouží i jako šablona, pokud uživatel nemá

[3] Memoizace je technika optimalizace výpočtu, kdy se ukládají výsledky dříve provedených výpočtů a následně se tyto výsledky použijí místo opakování výpočtu. Takto se snižuje časová náročnost výpočtů, které mají stejný výsledek. V Reactu lze memoizovat výpočty pomocí hooku useMemo, které se přepočítají jen na základě poskytnutých závislostí. Pro memoizování funkcí slouží hook useCallback, čímž se zachová instance funkce mezi jednotlivými překresleními komponenty a minimalizují se nepotřebná překreslení dětských komponent. Ke změně funkce dochází pouze na základě změny poskytnutých závislostí.

sběrná místa ve svém sběrném systému. Excel obsahuje list se sběrnými místy (pokud existují) a list, ve kterém jsou popsané jednotlivé sloupce v prvním listu a jakými daty by měly být vyplněny i s příklady.

Funkce na sdružování tříd

Komponentám lze předávat třídy v rámci props, aby se upravili aktuální potřebě na daném místě. Proto se vytvořila jednoduchá funkce, která sdružuje definované třídy v komponentě a třídy předané pomocí props. Funkce také podporuje podmíněné nastavení třídy, což znamená, že třída je nastavena na základě splnění podmínky.

Světlý a tmavý režim aplikace

Jedná se o funkci, která si z lokálního úložiště načte hodnotu aktuálně vybraného barevného režimu. Na základě této hodnoty se z objektu, ve kterém jsou vydefinované CSS proměnné pro každý z režimů, načtou klíče a hodnoty a nastaví se v DOMu. Komponenty následně využívají tyto proměnné, čímž je docíleno možnosti měnění barevného režimu aplikace.

5 ZÁVĚR

Cílem této diplomové práce je návrh a vytvoření webové aplikace pro plánování rozmístění sběrných míst odpadu. V úvodu práce je uvedena motivace propojení závěrečných prací a výzkumných projektů v nástroj nazvaný Popelka, který slouží k plánování sběrné a svozové infrastruktury v odpadovém hospodářství a bude poskytnut jakožto podpůrný nástroj pro obce, kterým umožní efektivně navrhnout sběrnou a svozovou infrastrukturu. V rámci úvodu jsou uvedeny i cíle aplikace, které by měli být v této diplomové práci naplněny. Další kapitolou je obeznámení s odpadovým hospodářstvím České republiky, za kterou následuje kapitola s přehledem použitých technologií a kapitola popisující samotnou aplikaci.

Před samotným vývojem aplikace bylo potřeba ve spolupráci s dalším studentem vytvořit návrh aplikace a vybrat technologie, které se při vývoji použijí. V rámci návrhu aplikace bylo potřeba promyslet i komunikaci s výpočetními jádry, která byla vytvořena v rámci dalších závěrečných prací. Následně během vývoje docházelo k rozšiřování aplikace o další části a funkcionality, nad kterými se během fáze návrhu neuvažovalo.

Výsledná aplikace nabízí uživatelům přívětivé prostředí, ve kterém si mohou plánovat svou sběrnou infrastrukturu. Pro správu infrastruktury jsou v aplikaci vytvořeny dva moduly. V rámci prvního modulu si může uživatel vytvářet nebo generovat sběrná místa, která jsou seskupena do sběrného systému. Sběrná místa se zobrazují v mapě, kde je s nimi možnost dalších operací. V rámci druhého modulu je vytvořen přehled všech uživatelem vytvořených sběrných systémů, u kterých má možnost je zobrazit nebo smazat.

Aplikace byla vyvíjena s primárním cílením zobrazování na monitorech a větších tabletech. Na základě zpětné vazby uživatelů by jedním z dalších kroků vývoje aplikace mohla být podpora zobrazení na mobilních zařízeních. Jedním z plánovaných vylepšení je pravidelná aktualizace, alespoň jednou za rok, všech adresních míst dle aktuálního balíčku adresních míst z Českého statistického úřadu, který ovšem bude muset být zakoupen. Dalším z plánovaných rozšíření je výpočet a zobrazení chodníkových tras mezi sběrnými místy a adresními místy, čímž se zpřesní výpočet a následná statistika docházkové vzdálenosti od adresních míst ke sběrným místům.

I přes dosažení cílů stanovených v úvodu práce, vývoj aplikace není ukončen a pokračují práce na přidání dalších modulů. Jedná se o perspektivní aplikaci v oblasti odpadového hospodářství, která už má své první zájemce. Od uživatelů se tak začíná dostávat zpětná vazba a náměty na vylepšení a rozšíření funkcionalit aplikace, dle jejich praktických zkušeností.

SEZNAM POUŽITÉ LITERATURY

[1] Khýr, L. *Matematické modely v oblasti strategického rozhodování.* Brno, 2020. 69 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav matematiky. Vedoucí práce Ing. Martin Pavlas, Ph.D. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/121551.

[2] Nevrlý, V., Šomplák, R., Smejkalová, V., Lipovský, T. a Jadrný, J. Location of municipal waste containers: Trade-off between criteria. *Journal of Cleaner Production.* 2021, sv. 278, s. 123445. DOI: https://doi.org/10.1016/j.jclepro.2020.123445. ISSN 0959-6526. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0959652620334909.

[3] Silný, M. *Analýza rozmístění sběrných míst a kontejnerů pro komunální odpady.* Brno, 2022. 61 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav procesního inženýrství. Vedoucí práce Ing. Vlastimír Nevrlý, Ph.D. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/140674.

[4] TIRSMZP719. *Prognózování produkce odpadů a stanovení složení komunálního odpadu* [online]. [cit. 2023-04-05]. Poskytovatel: Technologická agentura České republiky. Doba řešení: 1.1.2019 – 30.6.2022. Dostupné z: https://www.vut.cz/vav/projekty/detail/30096.

[5] Kubowský, J. *Webová aplikace pro plánování svozu frakcí odpadu.* Brno, 2022. 62 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce: Ing. Ladislav Dobrovský. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/139983.

[6] Ministerstvo životního prostředí. *Odpadové hospodářství* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mzp.cz/cz/odpadove_hospodarstvi.

[7] Ministerstvo životního prostředí. *Strategický rámec cirkulární ekonomiky České republiky 2040* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mzp.cz/cz/cirkularni_cesko.

[8] Ministerstvo životního prostředí. *Plnění nařízení vlády o Plánu odpadového hospodářství ČR* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mzp.cz/cz/plneni_narizeni_vlady.

[9] Ministerstvo životního prostředí. *Odpady* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mzp.cz/cz/odpady_podrubrika.

[10] Ministerstvo životního prostředí. *Plán odpadového hospodářství ČR* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mzp.cz/cz/plan_odpadoveho_hospodarstvi_cr.

[11] Ministerstvo životního prostředí. *Zpětný odběr výrobků* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mzp.cz/cz/zpetny_odber_vyrobku.

[12] Ministerstvo průmyslu a obchodu. *Balíček k oběhovému hospodářství* [online]. [cit. 2023-04-05]. Dostupné z: https://www.mpo.cz/cz/stavebnictvi-a-suroviny/strategicke-dokumenty-pro-udrzitelne-stavebnictvi/balicek-k-obehovemu-hospodarstvi--173269/.

[13] Boskovic, G. a Jovicic, N. Fast methodology to design the optimal collection point locations and number of waste bins: A case study. *Waste Management & Research*. 2015, sv. 33, č. 12, s. 1094–1102. DOI: 10.1177/0734242X15607426. Dostupné z: https://doi.org/10.1177/0734242X15607426.

[14] Hemmelmayr, V., Doerner, K., Hartl, R. a Vigo, D. Models and Algorithms for the Integrated Planning of Bin Allocation and Vehicle Routing in Solid Waste Management. *Transportation Science*. Leden 2012, sv. 48, s. 103–120. DOI: 10.1287/trsc.2013.0459. Dostupné z: https://doi.org/10.1287/trsc.2013.0459.

[15] Rossit, D. G., Toutouh, J. a Nesmachnow, S. Exact and heuristic approaches for multi-objective garbage accumulation points location in real scenarios. *Waste Management*. 2020, sv. 105, s. 467–481. DOI: https://doi.org/10.1016/j.wasman.2020.02.016. ISSN 0956-053X. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0956053X2030074X.

[16] *Flask* [online]. [cit. 2023-04-05]. Dostupné z: https://flask.palletsprojects.com/en/2.2.x/.

[17] *Werkzeug* [online]. [cit. 2023-04-05]. Dostupné z: https://palletsprojects.com/p/werkzeug/.

[18] *MariaDB* [online]. [cit. 2023-04-05]. Dostupné z: https://mariadb.org/.

[19] *NGINX* [online]. [cit. 2023-04-05]. Dostupné z: https://www.nginx.com/.

[20] *UWSGI* [online]. [cit. 2023-04-05]. Dostupné z: https://uwsgi-docs.readthedocs.io/en/latest/.

[21] *Redis* [online]. [cit. 2023-04-05]. Dostupné z: https://redis.io/.

[22] *Redis Queue* [online]. [cit. 2023-04-05]. Dostupné z: https://python-rq.org/.

[23] *React* [online]. [cit. 2023-04-05]. Dostupné z: `https://reactjs.org/`.

[24] *Redux* [online]. [cit. 2023-04-05]. Dostupné z: `https://redux.js.org/`.

[25] *React Router* [online]. [cit. 2023-04-05]. Dostupné z: `https://reactrouter.com/en/main`.

[26] *OpenStreetMap* [online]. [cit. 2023-04-05]. Dostupné z: `https://www.openstreetmap.org/about`.

[27] *Leaflet* [online]. [cit. 2023-04-05]. Dostupné z: `https://leafletjs.com/`.

[28] *React Leaflet* [online]. [cit. 2023-04-05]. Dostupné z: `https://react-leaflet.js.org/docs/start-introduction/`.

[29] *SheetJS* [online]. [cit. 2023-04-05]. Dostupné z: `https://sheetjs.com/`.

[30] *Recharts* [online]. [cit. 2023-04-05]. Dostupné z: `https://recharts.org/en-US/`.

[31] *Ajax* [online]. [cit. 2023-04-05]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX`.

SEZNAM ZKRATEK A SYMBOLŮ

AJAX	Asynchronous JavaScript and XML
API	Aplication Programming Interface
BP	Bakalářská práce
CSS	Cascading Style Sheets
ČR	Česká republika
DB	Database
DOM	Document Object Model
DP	Diplomová práce
ER	Entity Relationship
EU	Evropská unie
FIFO	First In, First Out
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IMAP	Internet Message Access Protocol
JSON	JavaScript Object Notation
JSX	JavaScript XML
ORP	Obec s rozšířenou působností
POH ČR	Plán odpadového hospodářství České republiky
POP3	Post Office Protocol
SKO	Směsný komunální odpad
SMTP	Simple Mail Transfer Protocol
SPA	Single-page application
SQL	Structured Query Language

SSL Secure Socket Layer

SVG Scalable Vector Graphics

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UI User Interface

URL Uniform Resource Locator

UX User Experience

ÚPI Ústav procesního inženýrství

WSGI Web Server Gateway Interface

XHTML EXtensible HyperText Markup Language

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

ZOV Zpětný odběr výrobků

SEZNAM OBRÁZKŮ