Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Webová aplikace „Bin finder”

Bakalářská práce

Oleksandr Bondarenko

Vedoucí práce: PhDr. Miloš Prokýšek, Ph.D.

České Budějovice 2021

Bondarenko O., 2021: Webová aplikace „Bin finder". [Web application „Bin finder". Bc. Thesis, in Czech] – 47 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Bakalářská práce se zabývá vytvořením webové aplikace *„Bin finder"*, která bude poskytovat uživateli příslušná data o poloze kontejnerů, určených pro tříděný komunální odpad. V rámci této práce je vyvíjena aplikace, která bude realizována formou jednostránkové webové stránky. Součástí práce je zadávací dokumentace, analýza funkčních a nefunkčních požadavků, dokumentace kódu, testovací a uživatelská dokumentace.

Annotation

The bachelor thesis deals with the creation of web application *"Bin finder"*, which will provide the users with relevant data on the location of containers intended for sorted municipal waste. The application will be developed in the form of a SPA (Single Page Application). Part of the work is tender documentation, analysis of functional and non-functional requirements, code documentation, test and user documentation.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 12. 02. 2021 Oleksandr Bondarenko

Poděkování

Rád bych poděkoval panu PhDr. Miloši Prokýškovi, Ph.D., za jeho čas strávený při konzultacích a velmi cenné rady. Dále bych rád poděkoval své manželce paní Ing. Viktorii Petrenko za její podporu během studia.

Obsah

1 Úvod

Mnou zvolené téma bakalářské práce zahrnuje vytvoření softwarového řešení pro vyhledávání volně přístupných kontejnerů pro tříděný komunální odpad, které by mělo poskytnout uživateli informace o umístění kontejnerů na základě uživatelem přednastavených voleb (adresa, typ odpadu apod.). Vyvíjená aplikace je zaměřena na zprostředkovávání dat formou portálu a bude obsahovat data o nádobách na území hlavního města Prahy, které budou přístupné jak pro obyvatelstvo, tak i pro návštěvníky města.

Z uživatelského hlediska by vyvíjená aplikace měla zároveň plnit zájem uživatele o požadované informace a poskytovat prosté a zároveň příjemné uživatelské rozhraní.

Tato práce se zabývá vytvořením klientské a serverové části webové aplikace, grafického návrhu responzivního uživatelského rozhraní, uživatelské a testovací dokumentace.

Hlavním důvodem vzniku tohoto projektu je usnadnit obyvatelstvu a návštěvníkům hlavního města Prahy vyhledávání kontejnerů na tříděný odpad. Dalším důvodem je vytvoření aplikace, která by umožňovala snadné rozšíření i na další města ČR, a tím pádem by bylo možné podstatně zjednodušit vyhledávání kontejnerů například při cestování napříč celou ČR bez zbytečného pamatování si jednotlivých místních portálů.

Prvním krokem při zpracování této bakalářské práce bylo provedení analýzy podobných softwarových řešení, která umožnila specifikovat výhody jednotlivých portalů. Pak následovala definice logického rámce prace, funkčních a nefunkční požadavků, na základě výstupů z předchozích kapitol byly definovány uživatelské scénáře a navržen grafický design uživatelského rozhraní aplikace. V dalším kroku probíhalo vyhledávání a volba základních knihoven projektu a datových zdrojů aplikace. Dalším krokem bylo navržení architektury aplikace a následná implementace projektu. Dále následovalo nasazení aplikace do testovacího provozu, kde se důkladně testovaly. V rámci této práce byl proveden úspěšný pokus o rozšíření aplikace na další město Ostravu.

Finálním krokem bylo provedení zhodnocení a formalizace výstupů projektu.

1.1 Cíle práce

Hlavním cílem práce je vytvoření webové aplikace *„Bin finder"*, která poskytne uživateli informace o sběrných nádobách pro tříděný komunální odpad v jeho blízkosti.

Aplikace bude získávat a zobrazovat data z existujících OpenData zdrojů ve formě interaktivní mapy, s možností zobrazení podrobných informací.

Součástí práce budou bezpodmínečně následující dokumenty:

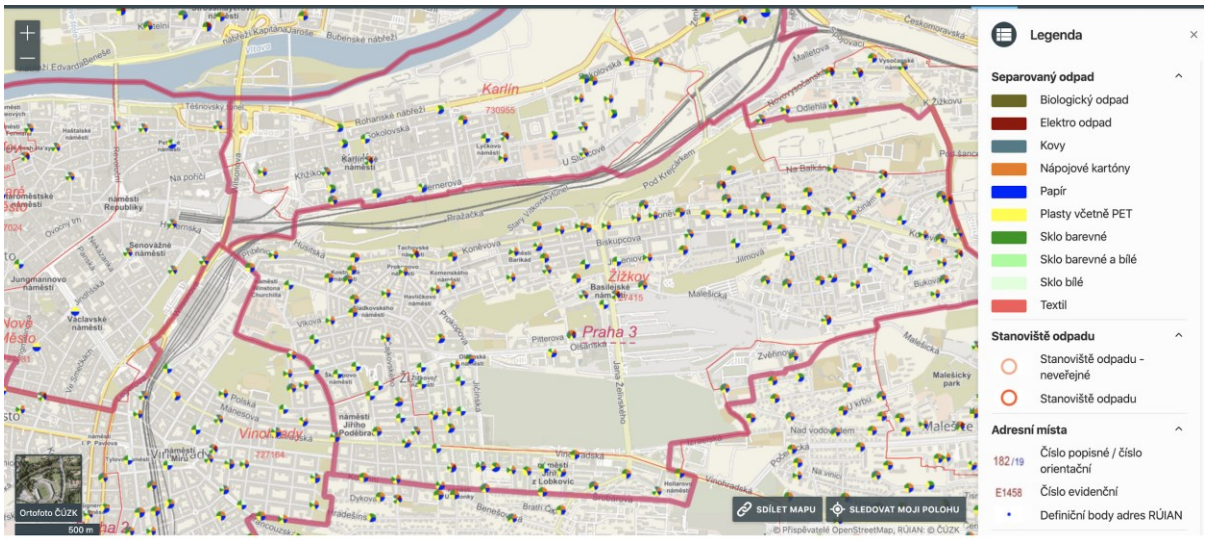
- Zadávací dokumentace
- Analýza funkčních a nefunkčních požadavků
- Dokumentace kódu
- Testovací dokumentace
- Uživatelská dokumentace

2 Analýza

2.1 Analýza podobných softwarových řešení

V současné době je na území České republiky v provozu několik portálů, poskytujících informace o umístění nádob na tříděný komunální odpad. První z naznačených je *„Mapa tříděného odpadu"* [1], který je provozován na území hlavního města Prahy, tento portál poskytuje velké množství podrobných informací o přístupných stanovištích tříděného odpadu. Navíc dává uživatelům možnost vyhledat potřebné stanoviště na základě zadané adresy, avšak z hlediska UI/UX naznačený portál vizualizuje přebytečné informace např. identifikační číslo kontejneru, dodatečné vrstvy na mapě apod. Uživatelem požadovaná data jsou de facto „promíchaná" s technickými daty, ve výsledku je uživatel musí zbytečně vyhledávat v zobrazených výsledcích. Uživatelské rozhraní portálu je znázorněno na obrázku 1.

Obrázek 1: Grafické rozhraní aplikace „Mapa tříděného odpadu" (hlavní město Praha)



Zdroj: [1]

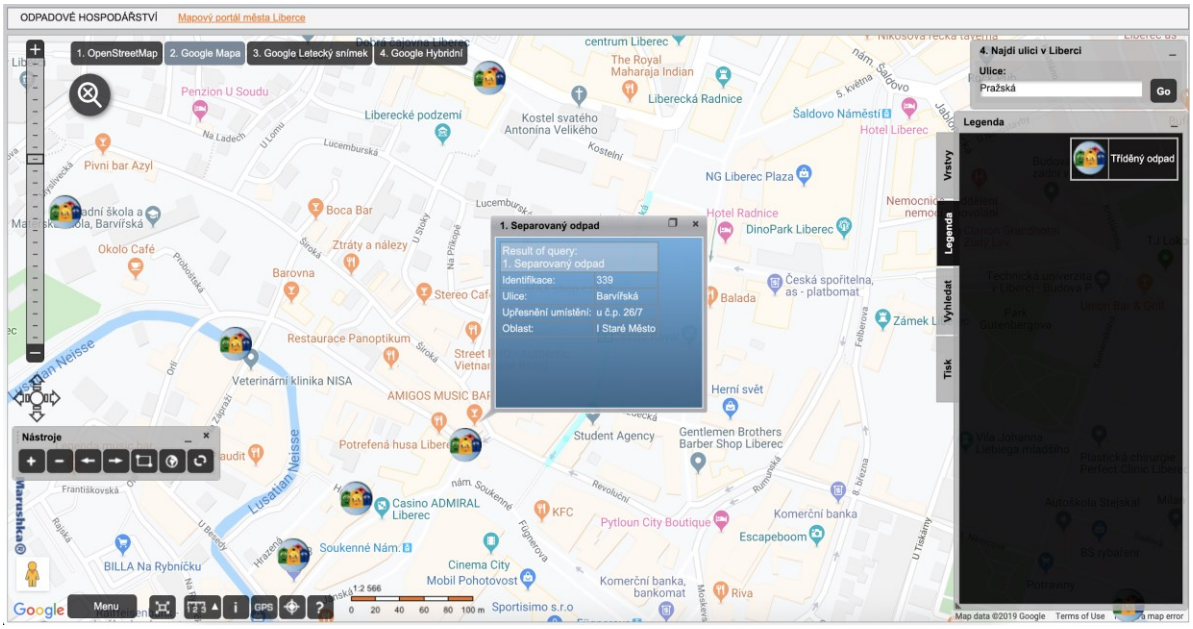
Na webových stránkách statutárního města České Budějovice lze najít odkaz na aplikaci *„Sběrná místa separovaného odpadu"* [2], která je implementovaná na žádost města společnosti *FCC Environment*. Aplikace zajišťuje vyhledávání nádob na území města a poskytuje možnost volby typu odpadu před vyhledáváním. Vyhledávání probíhá pomalu, občas během zpracování požadavku dochází k výpadkům aplikace. Ovládání uživatelského prostředí se provádí pomocí několika posuvných UI elementů a tlačítek. Tento portál dokáže vyhledat adresu a zobrazit frekvenci odvozu odpadu. Uživatelské rozhraní portálu je znázorněno na obrázku 2.

Obrázek 2: Grafické rozhraní aplikace „Sběrná místa separovaného odpadu" (České Budějovice)

Zdroj: [2]

Statutární město Liberec má na svém území nasazený do provozu integrovaný portál *„Úřad on-line"*, součástí kterého je *„Mapový portál"* [3] umožňující přístup k informacím o umístění nádob odpadového hospodářství města. Tento portál v porovnání s ostatními má rychlou odezvu. Díky využití *Google API*s portál umožňuje prohlížení ulic města a disponuje dobře známým uživatelům rozhraním *„Google Maps"*. Bezproblémově vyhledává jednotlivé ulice města, vyhledávání podle typu separovaného odpadu není k dispozici. Toto softwarové řešení je realizováno formou SPA. Grafické rozhraní daného portálu (Odpadové hospodářství) je znázorněno na obrázku 3.

Obrázek 3: Grafické rozhraní aplikace „Mapový portál (Odpadové hospodářství)" (statutární město Liberec)



Zdroj: [3]

Na základě výše uvedeného popisu existujících softwarových řešení a jejich uživatelského testování lze specifikovat výhody jednotlivých řešení. Výhody uvedené v tabulce 1.

Tabulka 1: Porovnání výhod webových portálů poskytujících informace o umístění nádob na tříděný komunální odpad provozovaných městy České republiky

	Název aplikace		
Srovnávací kritéria	*„Mapa tříděného odpadu" (České Budějovice)*	*„Mapový portál" (Liberec)*	*„Mapa tříděného odpadu" (hlavní město Praha)*
Možnost vyhledávání kontejneru na základě adresy	*Ano*	*Ne*	*Ano*
Stabilně rychlé zpracování požadavků uživatele	*Ne*	*Ano*	*Ano*
Implementace formou SPA	*Ne*	*Ano*	*Ano*
Aplikace vizualizuje optimální objem dat	*Ano*	*Ano*	*Ne*
Příjemné uživatelské rozhrání	*Ne*	*Ano*	*Ano*
Aplikace umožňuje uživatelům nastavovat vyhledávácí filtry	*Ano*	*Ne*	*Ne*

Zdroj: Vlastní zpracování

2.2 Logický rámec

Tabulka 2: Logický rámec projektu (1. část)

Popis projektu	Objektivně ověřitelné ukazatele	Prostředky ověření	Předpoklady
Záměr projektu: *Usnadnit obyvatelstvu hlavního města Praha vyhledávání kontejnerů pro tříděný komunální odpad*	*Funkční webová aplikace*	*Statistiky aplikace* *Hodnocení uživatelů*	
Cíl projektu: *Vytvořit interaktivní webovou aplikaci, která poskytne uživateli informace o sběrných nádobách pro tříděný komunální odpad v jeho blízkosti*	*Využívání aplikace obyvatelstvem města* *Více než 50 návštěv domény s aplikací během prvního provozního týdne*	*Statistiky návštěvnosti domény s aplikací*	*Uživatelský zájem o aplikaci*
Výstupy: *1. zadávací dokumentace* *2. webová aplikace* *4. uživatelská dokumentace* *5. testovací dokumentace*	*Zadávací dokumentace* *Webová aplikace nasazena do provozu* *Uživatelská dokumentace* *Testovací dokumentace*	*Do konce března 2020 bude navržen a implementován frontend aplikace.* *Do konce července 2020 bude provedena implementace backendu aplikace a její propojení s frontendem.* *Do konce února 2021 bude provedeno nasazení do provozu a testování aplikace.*	*Existence potřebných OpenData zdrojů.* *Existence potřebných REST APIs.* *Správné propojení využívaných open source balíků.* *Kompatibilita použitých open source balíků.*

Zdroj: Vlastní zpracování

Tabulka 3: Logický rámec projektu (2. část)

Aktivy projektu	Prostředky	Časový rámec aktivit	Předpoklady
Aktivy projektu: *1.1 analýza požadavků projektu* *1.2 rámcový návrh frontendu aplikace* *1.2 obecný návrh backendu aplikace* *2.1 implementace klientské části aplikace* *2.2 implementace serverní části aplikace* *2.3 propojení klientské a serverní části aplikace* *3. nasazení aplikace do testovacího provozu* *4.1 testování aplikace*	*Testovací lokální webový server* *Vývojové prostředí (IDE)* *Doména pro nasazení aplikací* *Webový server* *Webový prohlížeč*	***1.*** *1.1.2020* ***2.*** *1.5.2020* ***3.*** *1.8.2020* ***4.*** *1.1.2021*	*Dostatek časových a finančních prostředků*

Zdroj: Vlastní zpracování

2.3 Funkční požadavky

1. Aplikace bude získávat data o sběrných nádobách pro tříděný komunální odpad z openData zdrojů.
2. Aplikace bude schopna vizualizovat data ve formě interaktivní mapy.
3. Aplikace bude poskytovat uživateli informace o nádobách v jeho blízkosti.
4. Aplikace bude schopna zobrazit podrobné informace o nádobě.

2.4 Nefunkční požadavky

1. Aplikace s webovým rozhraní.
2. Aplikace implementovaná formou Single Page Application.
3. Minifikace datových toků klient-server.
4. Implementace responzivního designu aplikace.

2.5 Uživatelské scénáře

2.5.1 Uživatel chce vyhledat nádobu.

Základní stav: uživatel se nachází na úvodní obrazovce aplikace.

1. *Uživatel:* Stiskne tlačítko „***Vyhledat kontejnery na mapě***“ nebo otevře záložku ***„Vyhledat kontejnery“***.
2. *Systém:* Přesměruje uživatele na obrazovku ***„Interaktivní mapa”***.
3. *Uživatel:* Zadá adresu, v relativně které bude probíhat vyhledávání, dále nastaví uživatelské filtry (např. typ odpadu, poloměr vyhledávání).
4. *Uživatel:* Stiskne tlačítko ***„Vyhledat koše“***.
5. *Systém:* Zobrazí na mapě stanoviště tříděného odpadu, která se nacházejí v blízkosti uživatele nebo ukáže notifikační hlášku v případě nenalezení žádného kontejneru. V tomto případě scénář končí tímto bodem.
6. *Uživatel:* Klikne na nalezené stanoviště.
7. *Systém:* Zobrazí seznam nádob umístěných na vybraném stanovišti.
8. *Uživatel:* Klikne na libovolnou nádobu ze seznamu.
9. *Systém:* Zobrazí podrobné informace o nádobě.

2.5.2 Uživatel chce nechat zpětnou vazbu

Základní stav: uživatel se nachází na úvodní obrazovce aplikace.

1. *Uživatel:* Otevře záložku ***„Zpětná vazba”***.
2. *Systém:* Přesměruje uživatele na obrazovku ***„Formulář zpětné vazby”***.
3. *Uživatel:* Vyplní formulář a stiskne tlačítko ***„Odeslat”***.
4. *Systém:* Uloží uživatelem vyplněná data do databáze. Ukáže informační hlášku o úspěšném uložení dat. Přesměruje uživatele na hlavní obrazovku.

2.6 Front-end

Na základě osobních zkušeností s programovacím jazykem *JavaScript* a jeho nadstavbou *TypeScript* vydanou společností *Microsoft* bylo rozhodnuto na straně klienta používat jako programovací jazyk *TypeScript*.

2.6.1 Základní framework / knihovna

V současné době existuje velké množství nástrojů pro realizaci různorodých frontendových projektů v oblasti webu. Na základě předem definovaných funkčních a nefunkčních požadavků projektu je třeba zvolit vhodné nástroje pro implementační návrh frontendu vyvíjené webové aplikace.

Z výše specifikovaných funkčních a nefunkčních požadavků je zřejmé, že pro jejich splnění bude třeba použít vhodnou knihovnu nebo framework, který zajistí potřebnou interaktivitu a realizaci projektu formou SPA.

Mezi nejpoužívanější nástroje pro vytvoření podobných aplikací patří: *ReactJS*, *Angular*, *VueJS*. Jak je vidět z tabulky 4, každý z výše uvedených nástrojů má různou úroveň podpory a možnosti napojení open source balíků. Naznačený *Angular* se většinou používá pro velké projekty a má v sobě implementované velké množství různých modulů, které umožňují například validaci formulářů, komunikaci se serverem atd.

Na rozdíl od *Angularu*, *ReactJS* a *VueJS* nemají v sobě implementovanou tak různorodou funkcionalitu, avšak mají velké množství balíků a pluginů, které jsou k dispozici na https://www.npmjs.com/. V tomto případě vývojář sám rozhoduje, který balík bude využívat pro komunikaci se serverem, jak bude uchováván stav aplikace, jakým způsobem bude provedeno stylování uživatelského rozhraní aplikace apod.

Tabulka 4: Srovnání JavaScript frameworků / knihoven

	Framework / Knihovna		
Srovnávací kritérium	*ReactJS*	*Angular*	*VueJS*
Vydavatel	*Facebook*	*Google*	*Evan You (vývojář)*
Typ	*JavaScript knihovna*	*Framework*	*Framework*
Licence	*MIT*	*MIT*	*MIT*
Jazyk	*JavaScript, JSX*	*TypeScript*	*JavaScript*
Velikost minimalizovaného frameworku / knihovny	*~100 kB*	*~500 kB*	*~ 80 kB*
Podpora	*Facebook*	*Google*	*Open source community*
Možnost napojení open source balíků	*Ano*	*Ne (all in pack)*	*Ano*

Zdroj: [4]

Důležitým kritériem při výběru potřebného *JavaScript* frameworku / knihovny je jeho podpora vydavatelem. Jak je naznačeno v tabulce 4 *ReactJS*, je podporován společností *Facebook*, *Angular* je framework vydaný společností *Google*, u *VueJS* v tomto případě narazíme na podporu pouze autorem Evanem Younem a open source communitou, což může způsobit určitá rizika dalšího rozvoje aplikace např. změna API frameworku apod.

Pro zajištění vysoké míry spolehlivosti a eliminaci eventuálních rizik bylo rozhodnuto využít knihovnu ReactJS, která splňuje projektové požadavky a má několikrát menší velikost, která je důležitým parametrem pro první zobrazení webové aplikace, obzvlášť na mobilních zařízeních.

2.6.2 Funkcionální knihovny projektu

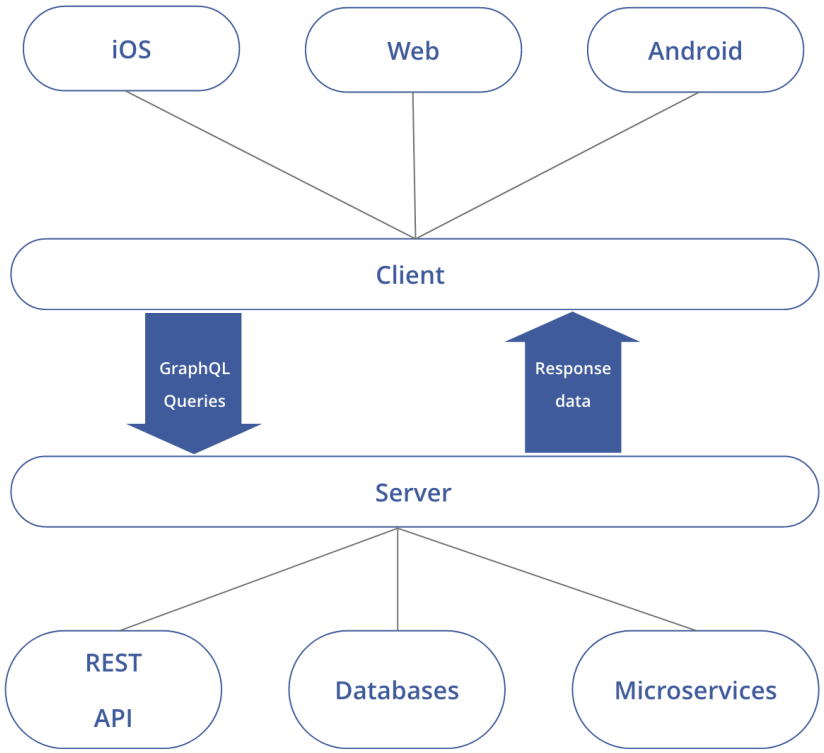
Jedním z nejdůležitějších úkolů v této fázi projektu je volba vhodných knihoven (balíků), které zajistí žádanou funkcionalitu webové aplikace. Na základě dříve zvolené *JavaScript* knihovny *ReactJS* a požadavků k projektu lze vyčlenit několik funkčních sfér, pro které by šlo nasadit vhodné balíky. Mezi ně patří:

- Organizace komunikaci mezi klientskou a severní částí aplikace
- Zajištění vizualizaci dat na interaktivní mapě

- Stylování uživatelského rozhraní aplikace

Pro organizaci komunikace mezi klientskou a severní částí aplikace je k dispozici několik dobře otestovaných a podporovaných balíků, které zajišťují komunikaci klient-server prostřednictvím REST API např. *Axios* [6], *Request* [7], avšak v dnešní době lze objevit i zcela odlišný přístup pro organizaci těchto datových toků, který je založen na samostatném dotazovacím jazyce GraphQL. Na rozdíl od technologie REST, GraphQL sjednocuje v sobě různé zdroje dat např. REST API, databáze apod. a poskytuje frontendu aplikace pouze jeden endpoint, na který se odesílají veškeré požadavky ze strany klienta, obrázek 4. Klientská část odesílá požadavek (tzv. query nebo mutation) na server a po určitém časovém intervalu dostane data v podobě JSON objektu. Technologie GraphQL umožňuje klientské části aplikace specifikovat požadovaná data deklarativním způsobem, a tím pádem dostávat je beze zbytečného vyčleňování. Specifikování datových struktur se provádí na serveru pomocí speciálního *„schéma"* souboru. Příkladem implementace technologií GraphQL pro *ReactJS* je balík: "react-apollo" [8].

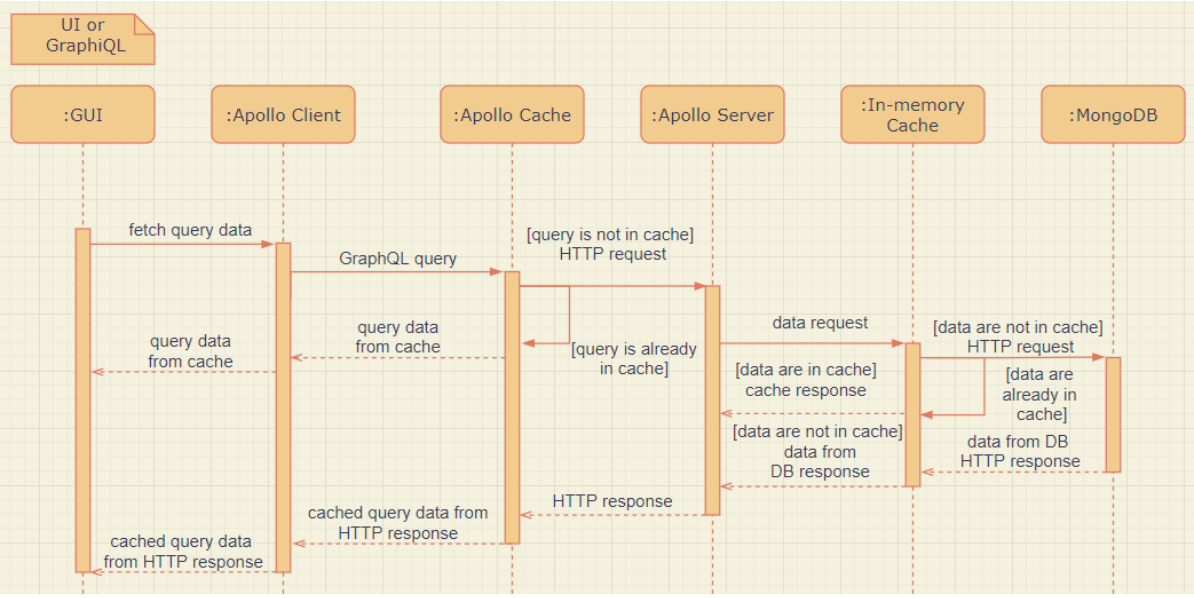
Obrázek 4: Schéma toku dat v Apollo GraphQL Environment



Zdroj: [5]

Výše zmíněný balík navíc disponuje normalizovanou klientskou cachi pro uchovávání dat ze serveru, což umožňuje předejít opakovanému odeslání požadavku na server, pokud požadovaná data jsou uložena do mezipaměti prohlížeče, obrázek 5 čímž lze docílit minifikace datových toků klient-server.

Obrázek 5: Sekvenční diagram případu Vyhledat kontejnery



Zdroj: Vlastní zpracování

Po porovnání jednotlivých výhod REST vs. *GraphQL*, pro zajištění komunikace klient-server pro webovou aplikaci *„Bin finder"* bylo rozhodnuto použít balík „react-apollo".

Integrace vizualizace dat ve formě interaktivní mapy s využitím *Google API* do knihovny *ReactJS* lze zajistit pomocí několika zcela podobných balíků např. „google-map-react" [10] nebo „react-google-maps" [11], které poskytují dobře známé uživatelské rozhraní *Google Maps*. Z analýzy stávajících řešení provedené v kapitole 2.1 je patrné, že využití servisů společnosti *Google* umožňuje realizovat rychlé, přívětivé a interaktivní uživatelské rozhraní. Pro implementaci interaktivní mapy separovaného komunálního odpadu byl vybrán balík „react-google-maps".

Stylování uživatelského rozhraní v prostředí *ReactJS* lze provést jak pomocí dobře známých preprocessorů kaskádových stylů (CSS) SCSS, SASS a LESS, tak i pomocí nativního CSS kódu. Obzvlášť pohodlným je použití speciálního balíku „styled-components", [12], jehož hlavním cílem je snadné stylování vyvíjeného produktu a eliminace duplicit stylů. Vzhledem k výše zmíněným výhodám balíku „styled-components", stylování aplikace *„Bin finder"* bude provedeno s využitím tohoto balíku.

2.7 Back-end

2.7.1 Webový server

Základním bodem pro definici backendového stacku aplikace je volba programovacího jazyka. Vzhledem k předdefinovanému frontendovému stacku aplikace, který bude implementován pomocí jazyka *JavaScript* (nadstavba *TypeScript*), snazším z hlediska vývoje je použití stejného jazyka i na serverové částí aplikace.

Dalším krokem je definice environmentálního prostředí backendu. V tomto případě, s ohledem na programovací jazyk, se dá zvolit prostředí *Node.js*, které tvoří *V8 engine* od společnosti *Google* a několik standardních knihoven. Navíc, na základě analýzy frontendu aplikace, provedené v kapitole 2.1, je možné vzít do úvahy existenci opensourcového softwarového řešení od společnosti *Apollo*, které de facto je nadstavbou frameworku *Express* od společnosti *Node.js*. Díky nasazení tohoto balíku bude možné docílit několika základních cílů:

- Minimalizace problémů optimalizace datových toků na straně serveru
- Ošetření interních chyb serveru
- Implementace GraphQL schématu

Pro implementaci serverové částí aplikace bylo rozhodnuto použít *Apollo Server*.

2.7.2 Databázový systém

Dnes se na trhu databázových systémů nabízí velké množství hotových řešení. Základním požadavkem na tento systém je možnost uchovávání potřebných datových struktur. S ohledem na strukturu používaných OpenData zdrojů, která obvykle je definovaná, ale může s časem změnit svou podobu, je vhodné použít NoSQL databázi. Jedním z dobře známých databázových systémů je *MongoDB*. Tento systém umožňuje relativně rychle a pohodlně měnit formu databázových záznamů, je bezplatným, a navíc zajišťuje rychlý a spolehlivý přístup k uloženým datům. Navíc *MongoDB* disponuje speciálním typem CRUD operace: *Geospatial Queries*, které umožňují vyselektování dat z databází pomocí speciálních geo queries, což umožňuje provádět vyhledávání dat efektivnějším způsobem.

Databázovým systémem aplikace *„Bin finder"* rozhodnuto použít *MongoDB*.

3 Implementace

V této kapitole jsou popsané jednotlivé implementační pokyny, které byly provedené během vývoje a které nejsou zřejmé z dokumentace kódu.

Vývoj aplikace Bin Finder bylo rozhodnuto rozdělit do dvou nezávislých repozitářů:

- Frontendová část – ***bin-finder-frontend***
- Backendová část – ***bin-finder-backend***

3.1 Datové zdroje aplikace

Jako základní datové zdroje aplikace byly zvoleny:

- OpenData sada „Stanoviště tříděného odpadu – položky“ (identifikátor: CZ-70883858-ZPK_CUR.ZPK_O_Kont_Toitem_b) [17]
- OpenData sada „Stanoviště tříděného odpadu“ (identifikátor: CZ-70883858-ZPK_CUR.ZPK_O_Kont_Tostan_b) [18]
- Google Places API (nápověda adres)
- Google Maps API (uživatelské rozhraní mapy aplikace)
- Web API (zdroj aktuální polohy uživatele)

Data z OpenData zdrojů byla agregována a umístěna do databáze, na základě těchto zdrojů byly vytvořeny dvě kolekce datových záznamů:

- Kolekce „waste_stations“ obsahuje záznamy o stanovištích na tříděný odpad v podobě JSON objektů, obrázek 6.

  Atributy záznamu:

  _id – identifikátor stanoviště

  location – souřadnicová poloha stanoviště

  containers – pole kontejnerů umístěných v rámci daného stanoviště
- Kolekce „waste_containers“ obsahuje záznamy o kontejnerech na tříděný odpad v podobě JSON objektů, obrázek 7.

  Atributy záznamu:

  _id – identifikátor kontejneru

  location – souřadnicová poloha kontejneru

  wasteStationId – identifikátor stanoviště, kde je umístěn tento kontejner

  trashTypename – typ odpadu

cleaningFrequencyCode – četnost vývozu (1. hodnota - délka období, 2. hodnota - četnost vývozu)

containerType – typ kontejneru

Obrázek 6: Příklad JSON objektu - stanoviště na tříděný odpad


```
{
  "_id": 1,
  "location": {
    "type": "Point",
    "coordinates": [14.87654645, 50.89532313]
  },
  "containers": [
    {
      "_id": 1,
      "trashTypename": "glass"
    },
    {
      "_id": 2,
      "trashTypename": "paper"
    }
  ]
}
```

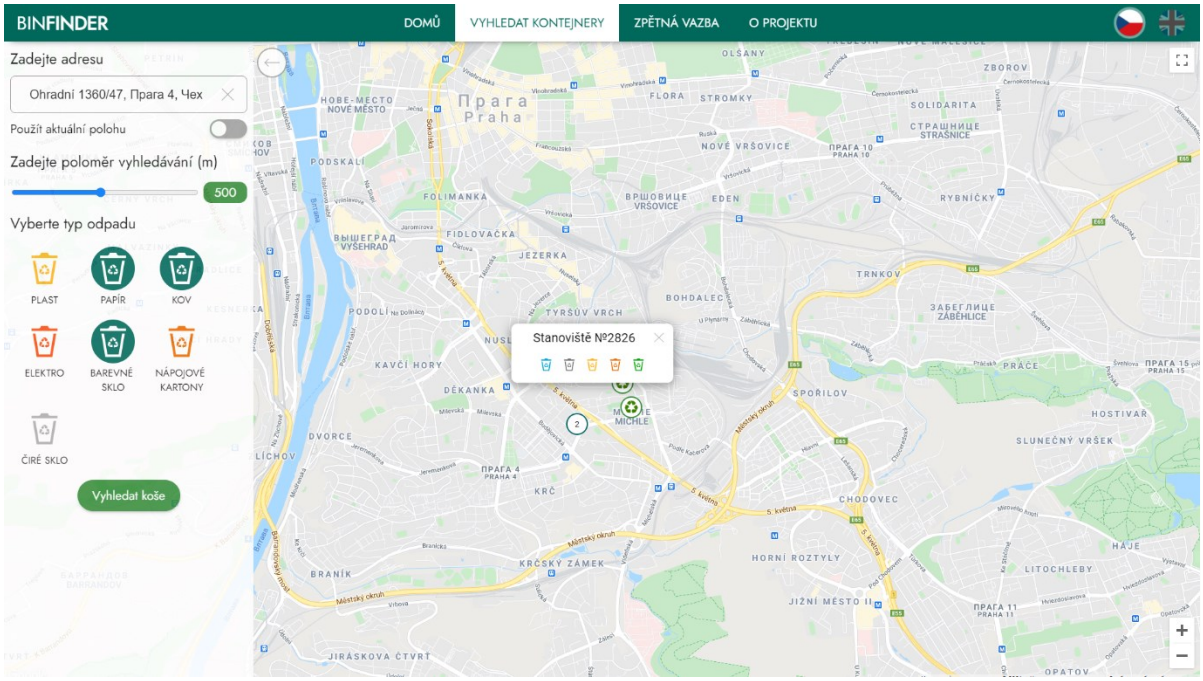

Zdroj: Vlastní zpracování

Obrázek 7: Příklad JSON objektu - kontejner na tříděný odpad


```
{
  "_id": 4,
  "location": {
    "type": "Point",
    "coordinates": [14.87654645, 50.89532313]
  },
  "wasteStationId": 1208,
  "trashTypename": "paper",
  "cleaningFrequencyCode": 41,
  "containerType": "3350 Atomium Reflex SV"
}
```


Zdroj: Vlastní zpracování

Obrázek 8: Ukázka uživatelského rozhraní aplikace „Bin Finder" (Stránka „Interaktivní mapa")



Zdroj: Vlastní zpracování

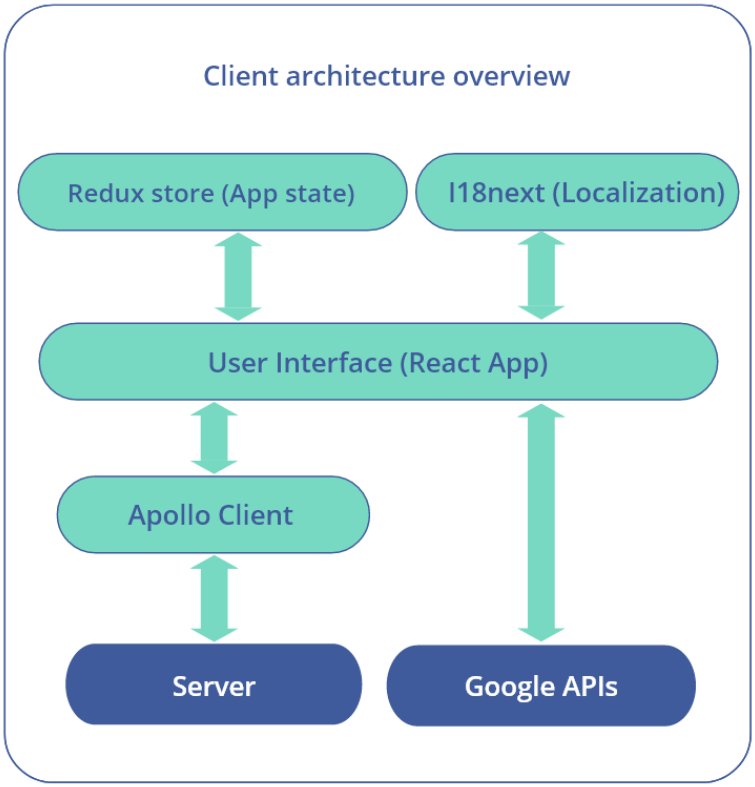
Při základním uživatelském scénáři (vyhledávání kontejnerů, obrázek 8) se datové toky v aplikaci využívají následujícím způsobem:

Po otevření stránky na vyhledávání kontejnerů se spouští stahování mapy přes *Google Maps* API, po stáhnutí uživatel může iteragovat s dobře známým rozhraním mapy od společnosti *Google*. Prvním krokem ze strany uživatele je volba adresy, podle relativně které bude probíhat vyhledávání kontejnerů, v tomto kroku uživatel může začít zadávat adresu manuálně (v tomto případě pro nápovědu bude využité *Google Places API*), anebo zvolit automatické načítání aktuální polohy pomocí speciálního přepínače *„Použít aktuální polohu“*. Po zapnutí výše zmíněného přepínače se uživateli ukáže potvrzovací dialog pro povolení získání potřebných dat aplikací přes standardní Web rozhraní *„Navigátor“*, které umožňuje získávání dat aktuální polohy klienta. Druhým krokem ze strany uživatele je označení potřebných typů kontejnerů, které má aplikace vyhledat. Po označení a stisknutí tlačítka *„Vyhledat koše“* klientská část aplikace odešle požadavek na server s potřebnými vstupními daty (označené typy odpadů, polohu vyhledávání, poloměr vyhledávání) a jako odpověď obdrží data o nalezených stanovištích v záznamech DB. Pokud uživatel bude chtít zobrazit podrobná data o určitém kontejneru, po kliknutí na stanoviště se znovu odešle dotaz na server pro získání detailních dat pro požadovaný kontejner.

3.2 Frontendová část systému

3.2.1 Přehled architektury klientské částí systému

Obrázek 9: Přehled architektury klientské částí aplikace „Bin finder"



Zdroj: Vlastní zpracování

Jak je zobrazeno na obrázku 9, klientská část aplikace je sestavena z několika modulů, zejména:

Redux store

Modul pro uchovávání a synchronizaci stavu aplikací napříč klientskou částí systému. Instance tohoto modulu se inicializuje při nastartování aplikací a likviduje se po restartování, v daném systému se hlavně využívá pro uchovávání „loading" stavu, stavu notifikačních hlášek, uživatelem označených typů odpadů k vyhledávání apod. Však tento modul neslouží k uchovávání citlivých uživatelských údajů např. hesel apod. Komunikace tohoto modulu s jádrem aplikace (UI) se provádí na principech založených na základě *Flux* architektury.

Apollo Client

Modul pro nastartování *GraphQL operations* a management (stahování, obnovení, uchovávání, ukládání do mezipaměti) stažených dat ze serveru a zprostředkování jich

grafickému rozhraní aplikace. Obsahuje několik vrstev API (*Cache API*, *Core API*), které se aktivně využívají v tomto systému.

User Interface

Jádro systému, React aplikace, hlavním účelem je dynamické generování HTML šablon obnovení grafického rozhraní aplikací na základě obdržených vstupů (uživatelské pokyny, data ze serveru).

Google APIs

Seznam API, poskytovaných společností *Google* pro zobrazení, napovídání a mapování dat. V aplikaci se využívají:

- *Places API* – napovídání adres
- *Maps JavaScript API* – zobrazení a mapování dat na mapě

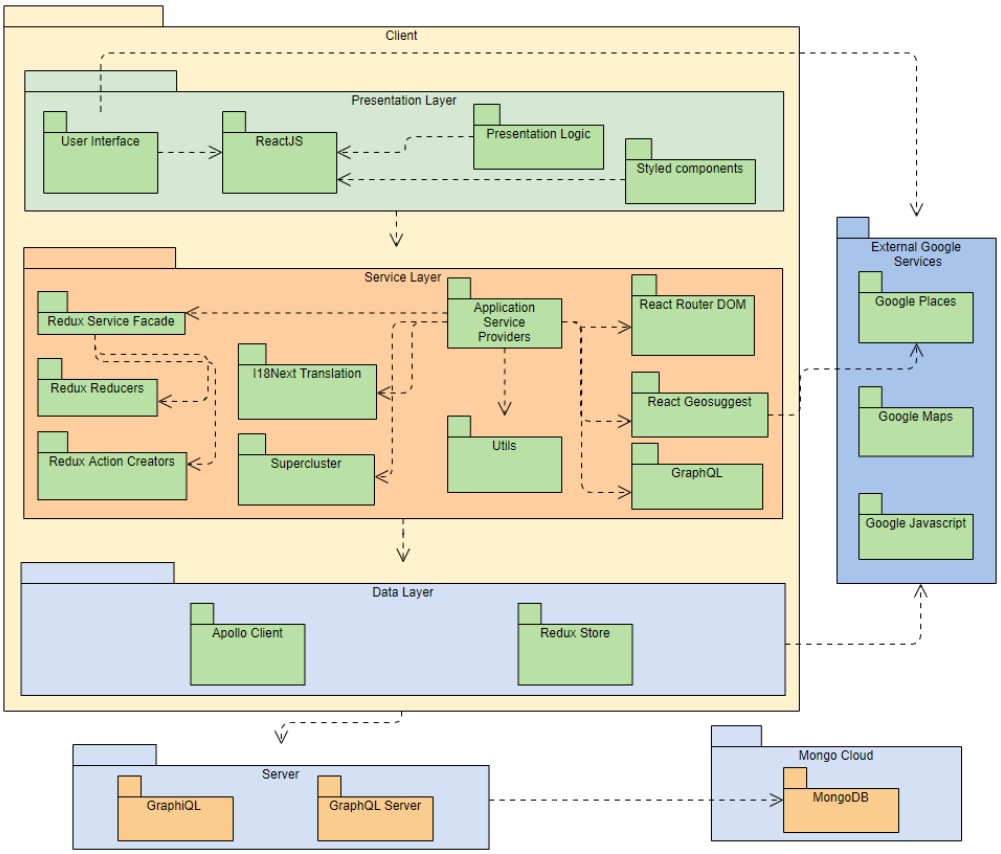
I18next

Modul, zajišťující lokalizaci uživatelského rozhraní. Disponuje vlastním API, díky kterému je provázán s jádrem systému.

Výše popsané moduly jsou rozdělené do tří aplikačních vrstev, obrázek 10:

- ***Presentation Layer*** – vrstva, která slouží pro zobrazení UI elementů a aktuálních dat uživateli
- ***Service Layer*** – aplikační vrstva, sloužící pro přenos a případnou transformaci dat mezi Data Layer a User Interface Layer
- ***Data Layer*** – vrstva aplikací sloužící především pro uchovávání dat

Obrázek 10: Package diagram aplikace ,,Bin finder"



Zdroj: Vlastní zpracování

3.2.2 Konfigurace klientské části systému

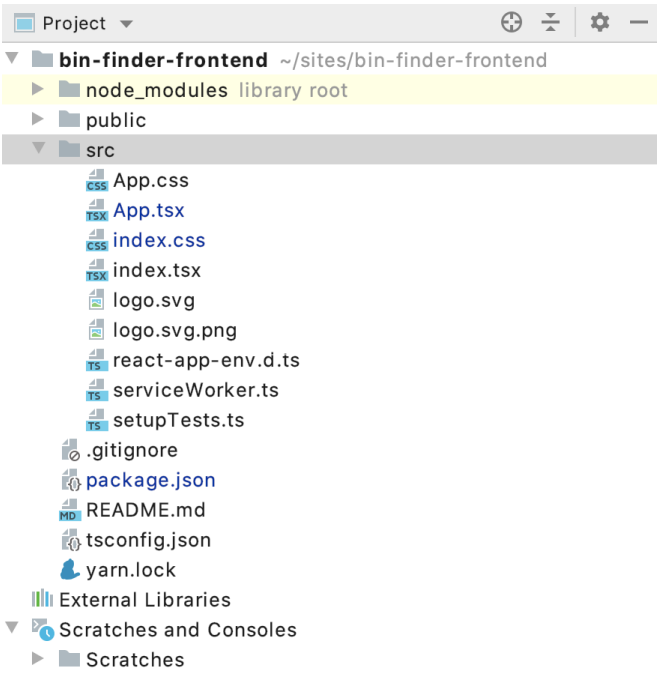
Konfigurační pokyny, které se týkají klientské částí projektu, jsou provedeny pomocí speciálního nástroje pro správu, stahování a spouštění balíků: *Yarn package manageru*.

V první řadě bylo třeba vytvořit *React* aplikací. Vytvoření a základní nastavení se provádí pomocí terminálového příkazu (CRA):

```
yarn create-react-app bin-finder-frontend --template typescript
```

Po provedení tohoto příkazu byla vygenerovaná defaultní struktura projektu, obrázek 11.

Obrázek 11: Defaultní struktura ReactJS projektu



Zdroj: Vlastní zpracování

Pomocí terminálového příkazu:

```
yarn add <package_name>
```

byla provedena instalace balíků, tabulka 5.

Pro dodržení principů psaní čistého kódu (TDD, DRY, SOLID) byla provedena instalace formatteru kodu *Prettier* a linteru *ESLint.*

Klientská část obsahuje následující seznam konfiguračních souborů:

- *.graphqlconfig* – konfigurace graphQL enginu
- *.prttierrc* – konfigurace formatteru kodu Prettier
- *.gitignore* – konfigurace nesledovaných souborů a složek projektu systémem Git
- *.codegen-graphql.yml* – konfigurační soubor pro generování *TypeScript* typu pro *GraphQL* operace
- *.codegen-scema.yml* – základní konfigurace zdroje, schéma souborů
- *tsconfig* – soubor pro konfiguraci *TypeScript*
- *package.json* – hlavní konfigurační soubor projektů, obsahuje popis a závislosti projektu

Odkazy na detailní popis konfiguračních souborů je umístěn v souboru *README.md*.

Tabulka 5: Balíky použité pro implementaci klientské části

Balík (součást produkčního bundlu)	Verze	Balík (pouze pro účely vývoje)	Verze
@apollo/react-hooks	*3.1.5*	*@graphql-codegen/cli*	*1.13.5*
@types/react-test-renderer	*16.9.3*	*@graphql-codegen/introspection*	*1.13.5*
apollo-boost	*0.4.7*	*@graphql-codegen/typescript*	*1.13.5*
google-map-react	*1.1.7*	*@graphql-codegen/typescript-operations*	*1.13.5*
graphql	*15.0.0*	*@graphql-codegen/typescript-react-*	
i18next	*19.5.0*	*apollo*	*1.13.5*
react-geosuggest	*2.12.1*	*@graphql-codegen/typescript-resolvers*	*1.13.5*
react-i18next	*11.7.0*	*@types/google-map-react*	*1.1.5*
react-redux	*7.2.0*	*@types/jest*	*25.1.4*
react-router-dom	*5.1.2*	*@types/react-geosugges*	*2.7.10*
styled-components	*5.0.1*	*@types/react-i18nex*	*8.1.0*
supercluster	*7.0.0*	*@types/react-redux*	*7.1.0*
use-supercluster	*0.2.8*	*@types/react-router-dom*	*5.1.3*
		@types/styled-component	*5.0.1*
		@typescript-eslint/eslint-plugin	*2.25.0*
		@typescript-eslint/parser	*2.25.0*
		eslint-config-prettier	*6.10.1*
		eslint-config-react-app	*5.2.1*
		eslint-plugin-react-hooks	*3.0.0*
		prettier	*2.0.2*

Zdroj: Vlastní zpracování

Detailní popis použitých balíků je součástí technické dokumentace projektu a je umístěn v souboru *README.md.*

Klientská část disponuje následujícími terminálovými příkazy:

- *yarn run start* – spouští aplikaci pro účely lokálního vývoje
- *yarn run build* – minifikue a vytváří produkční build aplikace
- *yarn run test* – skript spouští unit testy
- *yarn run codegen:schema* – spouští stahování *schema.json* souboru ze serveru
- *yarn run codegen:graphql* – spouští generování typu na základě staženého *schema.json* souboru

3.2.3 Přehled komponent

Do servisní vrstvy aplikace patří následující komponenty:

App

App je entry point pro *React* aplikace, který slouží pro propojení reálného a virtuálního DOMů webové aplikace. Obsahuje zprostředkovatele dat napříč celé aplikace (*AppoloProvider, ReduxProvider, I18NextProvider, GlobalStyles*).

AppLayout

Komponenta, která slouží pro rozmístění grafických prvků na webové stránce

Router

Zajišťuje navigaci mezi stránkami aplikací

Do UI vrstvy patří následující komponenty:

Dashboard

Obsahuje hlavní interakční prvky aplikace (volba typu odpadu, adresy vyhledávání, poloměru vyhledávání apod.).

Header

Navigační panel celé aplikace, umisťuje odkazy na jednotlivé stránky aplikace a přepínač jazyků uživatelského rozhraní.

Snackbar

Slouží pro zobrazení notifikačních hlášek aplikace.

Loading

Zobrazuje speciální loader, když se aplikace nachází ve stavu stažení dat ze serveru.

HomePage

Hlavní stránka aplikace, popisuje možné scénáře použití aplikace.

MapPage

Stránka pro vyhledávání kontejnerů pro tříděný odpad.

FeedbackPage

Stránka – formulář, pro zanechávání zpětné vazby uživatelem aplikace, možné návrhy pro vylepšení aplikací. Požadavky na umístění kontejnerů apod.

AboutPage

Stránka popisující hlavní účely projektu, proč tento projekt vznikl.

Aplikace *„Bin Finder“* obsahuje poměrně velké množství menších komponent pro zobrazování, stylování a další aktivity. Tyto komponenty jsou důkladně

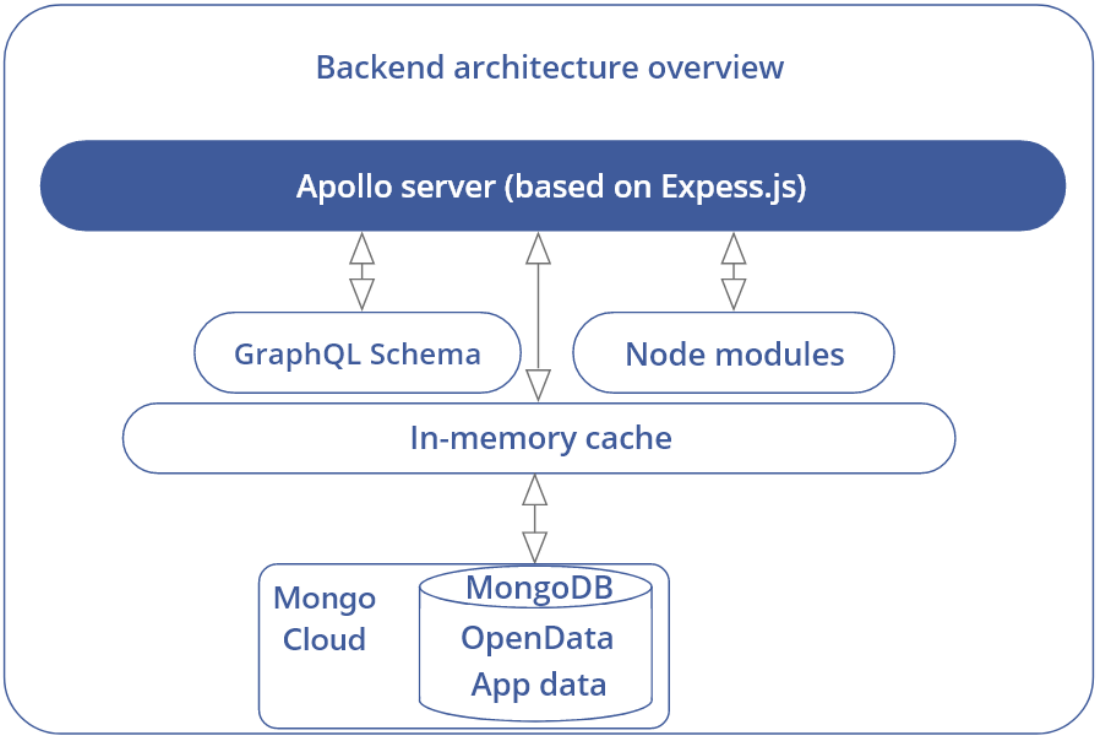
zdokumentované a popsané ve zdrojovém kódu projektu. Pro zjednodušení a předejití duplikování bylo rozhodnuto menší komponenty nepopisovat v textu práce.

3.3 Serverová část aplikace

3.3.1 Přehled architektury backendové části aplikace.

Jak bylo zmíněno v kapitole 2.7.1, jádro backendu aplikací tvoří GraphQL *Apollo Server*, který využívá ***schema.json*** jako soubor pro definici datových struktur, které mohou být vyžádané GraphQL klientem (*Apollo Client* v tomto případě). *Apollo Server* po obdržení požadavku z GraphQL klientovi spouští předem definovanou funkci (resolver), výsledkem které jsou klientem požadovaná data. Pro každou unikátní GraphQL operaci (query, mutace, subscription) se definuje samostatný resolver. V našem případě resolvery manipulují s daty uloženými v databázi *MongoDB*, která je umístěna na Cloudu. Komunikace s databází je zajištěna pomocí speciálního balíku ***mongoose***. Dotahovaná data se ukládají do mezipaměti na serveru pomocí standardní ***In-memory cache***, což umožňuje podstatně zmenšit odezvu serveru při dotahování klientem často využívaných dat, které nemohou být uložené do mezipaměti na straně klienta.

Obrázek 12: Přehled architektury backendové části aplikace „Bin finder"



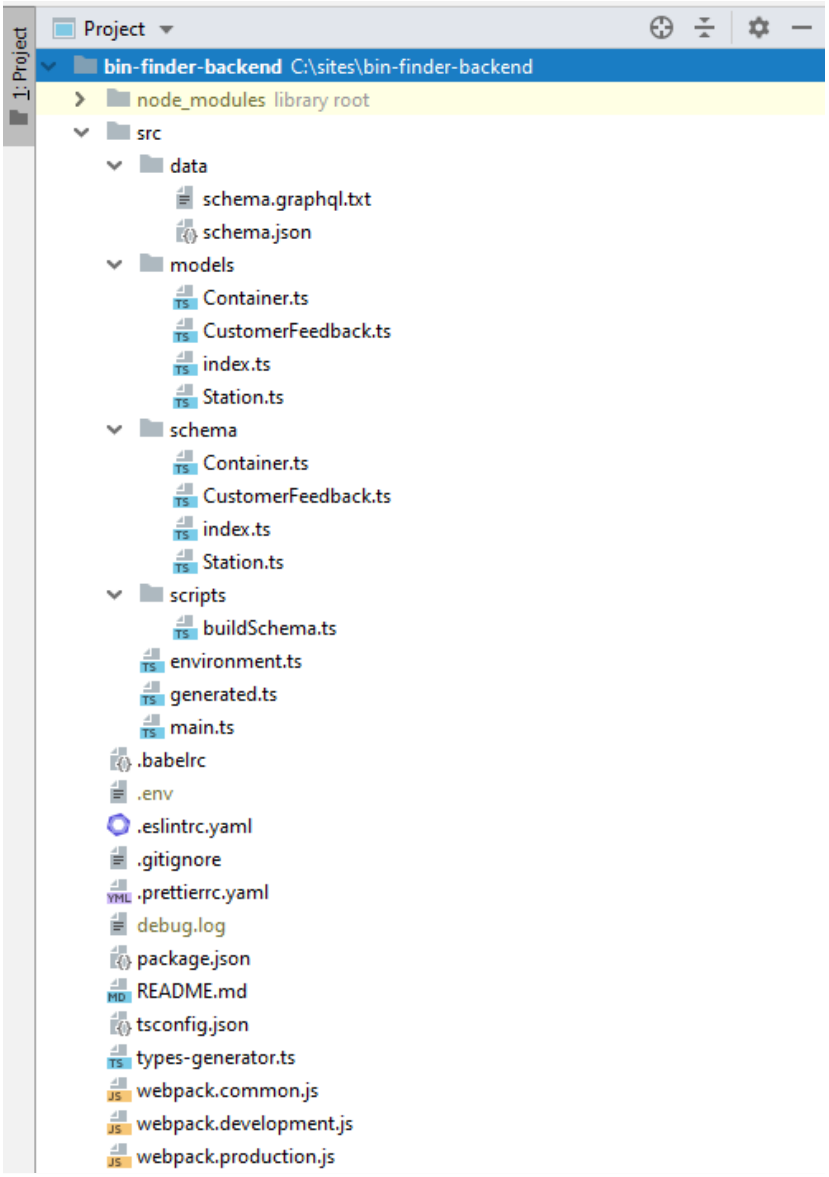
Zdroj: Vlastní zpracování

3.3.2 Struktura serverové části projektu

Jak je vidět na obrázku 13, serverová část aplikace *„Bin Finder“* se skládá z několika základních modulů:

- *data* – obsahuje vygenerovaný *schema.json* soubor, který bude vyžádán klientem
- *models* – obsahuje definici datových struktur
- *shema* – obsahuje *GraphQL* resolvery
- *scripts* – obsahuje služební scripty (generování *shema.json* souboru, generování TypeScript typů)

Obrázek 13: Struktura serverní části projektu „Bin Finder"



Zdroj: Vlastní zpracování

3.3.3 Konfigurace serverové části aplikace

Pro manipulaci s NPM balíky na serverové části, stejně jako na klientské, byl použit výše zmíněný *Yarn Package Manager*.

Pro zajištění stejného formátování kódu byl použit formatter *Prettier* a linter *ESLint*. Na rozdíl od klientské části, kde byla využita standardní konfigurace transpileru *Babel* a builderu *Webpack* (jsou defaultně využívány knihovnou *ReactJS* v rámci scriptu *CRA*) na backendu, bylo nutno definovat samostatné konfigurační soubory, obrázek 13:

- *.babelrc* – konfigurace transpileru *Babel*
- *webpack.commont.js* – konfigurace projektového builderu *Webpack* (sdílený soubor se využívá jak pro *development*, tak i pro *production* mode)
- *webpack.development.js* – konfigurace projektového builderu *Webpack* (*development* mode)
- *webpack.production.js* – konfigurace projektového builderu *Webpack* (*production* mode)
- *.prttierrc.yaml* – konfigurace formatteru kódu *Prettier*
- *.gitignore* – konfigurace nesledovaných souborů a složek projektu systémem Git
- *tsconfig.jcon* – soubor pro konfiguraci *TypeScript*
- *package.json* – hlavní konfigurační soubor projektu, obsahuje popis a závislosti projektu
- *.eslintrc.yaml* – konfigurace linteru *ESLint*

Odkazy na detailní popis konfiguračních souborů je umístěn v souboru *README.md* repositáře *bin-finder-backend.*

Detailní popis použitých balíků je součástí technické dokumentace projektu a je umístěn v souboru *README.md.*

Serverová část disponuje následujícími terminálovými příkazy:

- *yarn run start* – spouští Apollo Server pro účely lokálního vývoje
- *yarn run build* – minifikue a vytváří produkční build aplikace
- *yarn run generate-schema* – spouští generovaní *schema.json* souboru
- *yarn run generate* – spouští generování *TypeScript* typů na základě vygenerovaného *schema.json* souboru

Tabulka 6: Balíky, použité pro implementaci serverní části

Balík (součást produkčního bundlu)	Verze	Balík (pouze pro účely vývoje)	Verze
@babel/cli	*7.8.4*	*@types/mongoose*	*5.7.21*
@babel/core	*7.9.0*	*@types/webpack-env*	*1.14.1*
@babel/node	*7.8.7*	*@types/fs-extra*	*8.1.0*
@babel/preset-env	*7.9.0*	*babel-eslint*	*10.1.0*
apollo-engine	*1.1.2*	*babel-loader*	*8.1.0*
apollo-server	*2.10.1*	*babel-preset-env*	*1.7.0*
graphql	*14.6.0*	*clean-webpack-plugin*	*3.0.0*
graphql-compose	*7.16.1*	*dotenv*	*8.2.0*
graphql-compose-mongoose	*7.3.8*	*eslint*	*6.8.0*
mongoose	*5.9.15*	*eslint-plugin-babel*	*5.3.0*
		eslint-plugin-import	*2.20.1*
		eslint-plugin-node	*11.0.0*
		eslint-plugin-promise	*4.2.1*
		fs-extra	*9.0.0*
		graphql-schema-typescript	*1.3.2*
		graphql-tools	*4.0.7*
		nodemon	*2.0.2*
		prettier	*2.0.0*
		ts-loader	*6.2.0*
		ts-node	*8.6.2*
		typescript	*3.6.4*
		webpack	*4.41.2*
		webpack-cli	*3.3.9*
		webpack-merge	*4.2.2*
		webpack-node-externals	*1.7.2*

Zdroj: Vlastní zpracování

3.3.4 Spouštění serveru

Bodem vstupu serverové části apikace *„Bin Finder“* je soubor *main.ts*, ve kterém se provádí vytvoření instance *Apollo Serveru*, její spouštění a následující připojení do umístěné ve cloudu databázi *MondoDB*, obrázek 14. Pro účely lokálního vývoje se definuje soubor ***.env***, ve kterém jsou umístěna citlivá data. Z bezpečnostních důvodů tento soubor není součástí repozitáře, proto po naklonování projektu je nutné vytvořit tento soubor lokálně a vložit následující proměnné prostředí:

- ***MONGODB_URI****=mongodb+srv://bond****:<password>****@binfindercluster-medug.mongodb.net/bin_finder_db?retryWrites=true&w=majority*,
  (přístupové údaje k *MongoDB*, nutno vygenerovat vlastní URI)

- ***PORT**=8080*, (port, na kterém bude spuštěn server, pouze pro účely lokálního vývoje, na webovém serveru se inicializuje nezávislé tímto serverem)
- ***APOLLO_INTROSPECTION**=true*, (povolení využití GraphQL introspection)
- ***APOLLO_PLAYGROUND**=true*, (řídí zobrazení testovacího rozhraní *GraphiQL*)

Při nasazení do provozu musí být výše zmíněné proměnné definované na webovém serveru pomocí příslušných nástrojů poskytovatele servisu.

Obrázek 14: Ukázka fragmentů souboru main.ts

```
/**
 * Create Apollo Server
 */
const server = new ApolloServer( config: {
    schema,
    introspection: environment.apollo.introspection,
    playground: environment.apollo.playground,
    tracing: true,
    cors: true,
})

/**
 * Run Apollo Server
 */
server.listen( opts: { port: environment.port }).then( onfulfilled: ({ url }) => {
    // start connection to the DB
    const connection = mongoose.connect( uris: process.env.MONGODB_URI || '', options: {
        autoIndex: true,
        reconnectTries: Number.MAX_VALUE,
        reconnectInterval: 500,
        poolSize: 50,
        bufferMaxEntries: 0,
        keepAlive: true,
        useNewUrlParser: true,
    })

    mongoose.set('useCreateIndex', true)

    connection.then( onfulfilled: (db) => db).catch( onrejected: (err) => console.error(err))

    console.log(`Server listening on port ${url}`)
    console.log(`Health checks available at ${url}.well-known/apollo/server-health`)
})
```

Zdroj: Vlastní zpracování

4 Testování

4.1 Testování frontendu aplikace

Testování klientské části bylo provedeno s využitím unit, snapshot a manuálního testování. Pro zajištění automatického testování bylo rozhodnuto použít standardní testovací nástroje, které jsou součástí skriptu CRA, mezi ně patří:

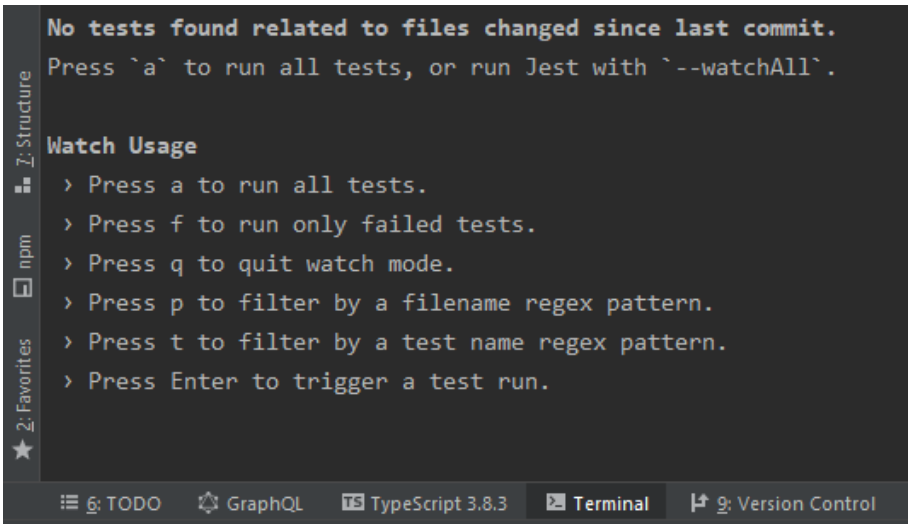
- *react-testing-library* – knihovna pro testování React komponent
- *Jest* – JavaScript Testing Framework

Veškeré unit testy se nachází v souborech s příponou ***.test.tsx***, což umožňuje bezproblémové rozeznání a spouštění testů pomocí příkazu:

```
yarn test
```

výsledkem tohoto příkazu je zobrazení speciálního terminálového dialogu, obrázek 15, pomocí kterého lze spustit jak veškeré testy v aplikaci, tak i pouze ty, které neprošly během posledního spouštění, také lze spustit speciální *„watch mode“*, během kterého *Jest* engine sleduje jednotlivé změny testu a automaticky spouští pouze modifikované testy. Tento přístup umožňuje vývojáři podstatně zrychlit proces psaní a opravy testů během vývoje.

Obrázek 15: Terminálový dialog testovacího frameworku Jest



Zdroj: Vlastní zpracování

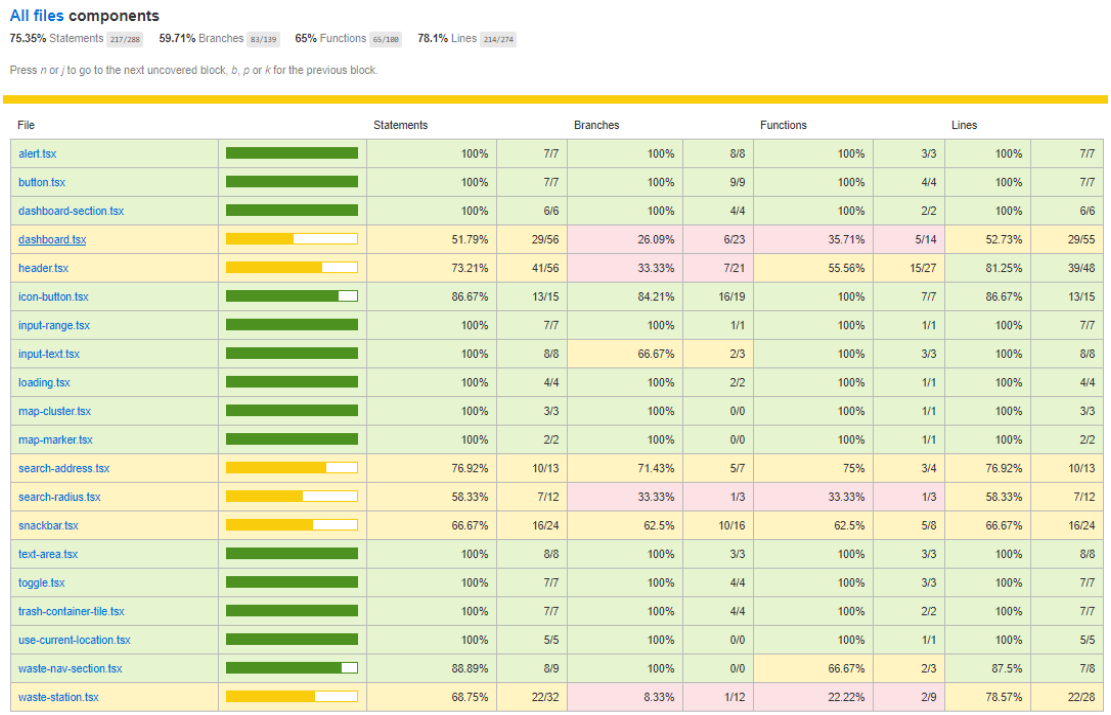
Na základě sepsaných testů lze vygenerovat speciální coverage report, díky kterému lze posoudit, jak dobře je tato aplikace pokrytá testy. Pro spouštění generování reportů je třeba spustit následující příkaz:

```
yarn test –coverage –watchAll=false
```

Jako výsledek se vygeneruje složka *coverage*, uvnitř které lze snadno vyhledat ***index.html*** soubor, jenž je třeba otevřít v libovolném webovém prohlížeči. Po otevření se zobrazí detailizovaný interaktivní test report, obrázek 16.

Unit a snapshot testy pokrývají více než 70 % klientské části aplikace.

Obrázek 16: Příklad test coverage reportu.



All files components

75.35% Statements 217/288 **59.71%** Branches 83/139 **65%** Functions 65/100 **78.1%** Lines 214/274

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

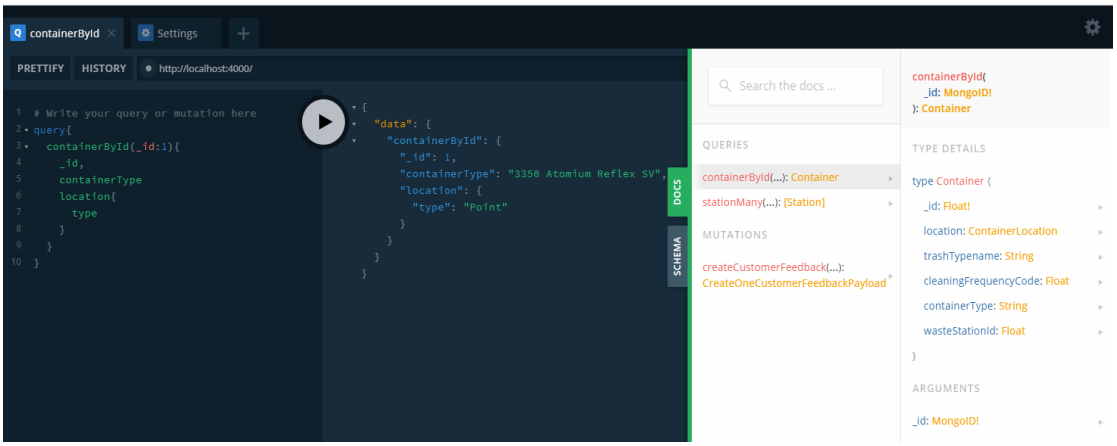
File	Statements		Branches		Functions		Lines	
alert.tsx	100%	7/7	100%	8/8	100%	3/3	100%	7/7
button.tsx	100%	7/7	100%	9/9	100%	4/4	100%	7/7
dashboard-section.tsx	100%	6/6	100%	4/4	100%	2/2	100%	6/6
dashboard.tsx	51.79%	29/56	26.09%	6/23	35.71%	5/14	52.73%	29/55
header.tsx	73.21%	41/56	33.33%	7/21	55.56%	15/27	81.25%	39/48
icon-button.tsx	86.67%	13/15	84.21%	16/19	100%	7/7	86.67%	13/15
input-range.tsx	100%	7/7	100%	1/1	100%	1/1	100%	7/7
input-text.tsx	100%	8/8	66.67%	2/3	100%	3/3	100%	8/8
loading.tsx	100%	4/4	100%	2/2	100%	1/1	100%	4/4
map-cluster.tsx	100%	3/3	100%	0/0	100%	1/1	100%	3/3
map-marker.tsx	100%	2/2	100%	0/0	100%	1/1	100%	2/2
search-address.tsx	76.92%	10/13	71.43%	5/7	75%	3/4	76.92%	10/13
search-radius.tsx	58.33%	7/12	33.33%	1/3	33.33%	1/3	58.33%	7/12
snackbar.tsx	66.67%	16/24	62.5%	10/16	62.5%	5/8	66.67%	16/24
text-area.tsx	100%	8/8	100%	3/3	100%	3/3	100%	8/8
toggle.tsx	100%	7/7	100%	4/4	100%	3/3	100%	7/7
trash-container-tile.tsx	100%	7/7	100%	4/4	100%	2/2	100%	7/7
use-current-location.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
waste-nav-section.tsx	88.89%	8/9	100%	0/0	66.67%	2/3	87.5%	7/8
waste-station.tsx	68.75%	22/32	8.33%	1/12	22.22%	2/9	78.57%	22/28

Zdroj: Vlastní zpracování

4.2 Testování serverové části

Testování backendu aplikace bylo zaměřeno především na testování jednotlivých GraphQL operací *Apollo Serveru*. Toto testování se provádělo manuálně jak během vývoje aplikace, tak i při každém deployment procesu. GraphQL endpoint disponuje relativně stručným API, proto bylo rozhodnuto provést izolované manuální testování v prostředí *GraphiQL* obrázek 17, kterým disponuje *Apollo Server*. Jak je vidět, rozhraní *GraphiQL* disponuje několika sekcemi. Vlevo se nachází editor pro psaní GraphQL operací, uprostřed se nachází kulaté tlačítko pro spouštění operace a sekce pro zobrazování výsledků volání. Vpravo toto rozhraní disponuje speciální záložkou ***DOCS*** po kliknutí, na kterou lze zobrazit seznam dostupných operací s možností zobrazení detailizovaného přehledu jednotlivé operace (veškerá data, která mohou být vyžádána od GraphQL serveru, specifikace datových typů apod.).

Obrázek 17: Rozhraní GraphiQL



Zdroj: Vlastní zpracování

5 Nasazení aplikace do pilotního provozu

Pro nasazení aplikací *„Bin Finder“* do pilotního provozu je třeba vyžít služeb poskytovatelů hosting servisu. Mezi nejpopulárnější poskytovatele těchto služeb patří *AWS*, *Microsoft Azure Portál* a mnoho dalších menších providerů. Hlavním kritériem při výběru hosting providera je v tomto případě bezplatné využití omezené funkcionality servisu postačující pro účely tohoto projektu.

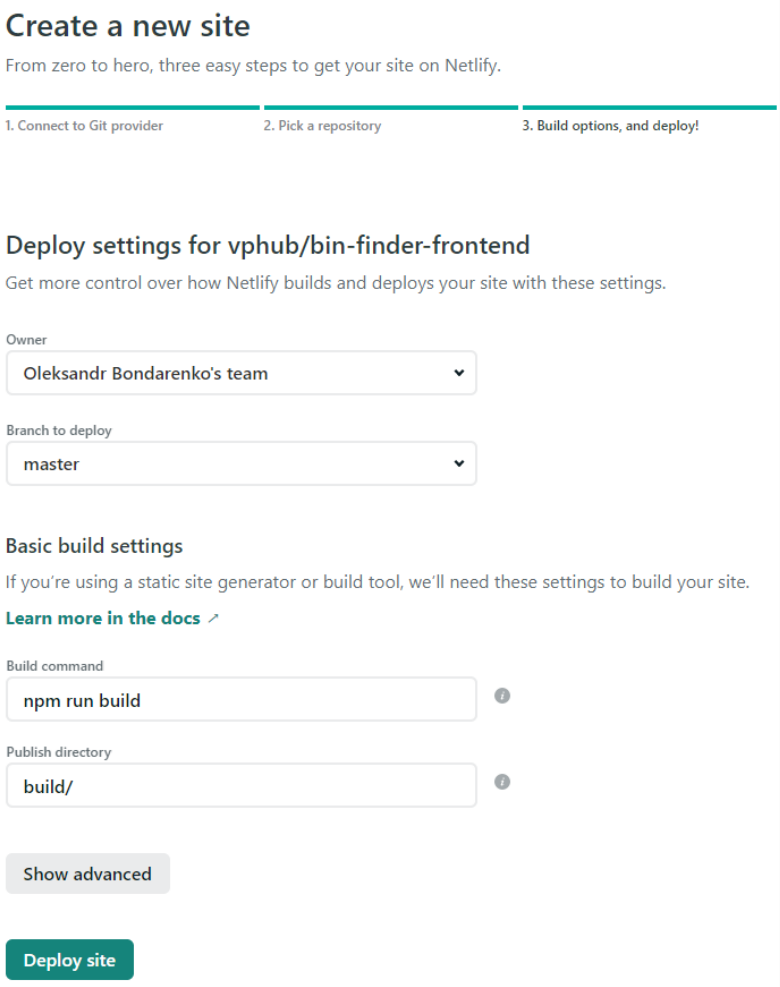
Po provedení srovnání jednotlivých poskytovatelů bylo zjištěno, že většina neposkytuje bezplatně tarify, avšak se podařilo vyhledat několik servisů, které splňují předem stanovené podmínky. Mezi ně patří dobře známé servisy Heroku a Netlify. Tyto servisy umožňují konfigurování automatického CI/CD procesů a nastavení proměnných prostředí, navíc servis *Netlify* umožňuje pozměnit název domény bezplatně (no human redable), a tím pádem ulehčit uživatelům zadávání URL adresy aplikace do webového prohlížeče během testovacího provozu. Tento servis splňuje potřeby klientské aplikace, avšak není postačující pro serverní část. Tento servis neposkytuje žádné prostředí pro rozběhání graphQL serveru. Toto omezení se dalo obejít pomocí rozdělení deployment prosesu klientské části pomocí servisu *Netlify* a serverní části pomocí servisu *Heroku*, který disponuje webovým serverem s prostředím *NodeJS*. Pokyny, popsané v rozdílech 5.1 a 5.2, vycházejí z předpokladu, že vývojář už má založené uživatelské účty v servisech *Heroku* a *Netlify*.

5.1 Deployment klientské části aplikace

Servis *Netlify* umožňuje propojit Git repozitář projektu, který může být umístěn ve službách *GitHub* nebo *Bitbucket* se svými interními servisy, Git repozitář klientské části aplikace už byl dříve umístěn do služby *Bitbucket*, tím pádem celý proces se skládal jen z vyplnění jednoduchého formu na stránkách app.netlify.com, obrázek 18.

Netlify využívá defaultní konfiguraci pro CI/CD proces, který umožňuje pohodlné změny pro vývojáře odesílat ze zdrojového kódu do vzdáleného *Bitbucket* Git repozitáře, následné interní servisy *Netlify* zachycují změny a provádí automatické nasazení nové verze aplikace.

Obrázek 18: Rozhraní pro deploy aplikace servisu Netlify



Zdroj: [14]

5.2 Deployment serverové části aplikace

Na rozdíl od *Netlify* servis *Heroku* nemá k dispozici propojovací nástroj pro napojení *Bitbucket* Git repozitáře, avšak má k dispozici speciální ***heroku*** CLI nástroj, který umožňuje provést nasazení naší aplikace na webový server pomocí terminálu. V první řadě je třeba stáhnout z oficiálních stránek [13] instalační program a nainstalovat nástroj *heroku* do operačního systému podle návodu programu. Po instalaci je nutné spustit speciální terminálový příkaz:

```
heroku login
```

Tento příkaz přesměruje uživatele na webovou stránku přihlášení do servisu *Heroku*. Pro vytvoření aplikací v servisu *Heroku* je třeba spustit další terminálový příkaz:

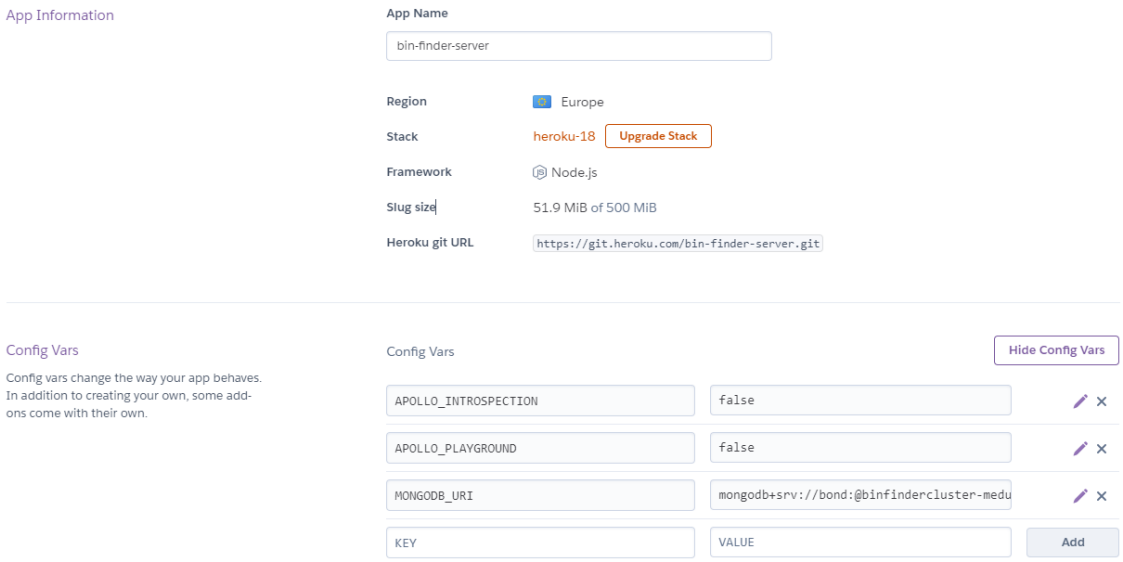
```
heroku create bin-finder-server,
```

který vytvoří prázdnou aplikaci s názvem *bin-finder-server* na webovém serveru. Dalším krokem je odeslání zdrojového kódu serverové části aplikace *„BinFinder“* do nově vytvořené aplikace na webovém serveru. Proto je třeba pomocí terminálu přejít do rootové složky projektu *bin-finder-backend* a spustit další příkaz:

```
git push heroku master
```

Po provedení tohoto příkazu se veškerý zdrojový kód odešle do interních servisů *Heroku*, pak tyto servisy spustí produkční verzi aplikací a spustí ji. Finálním krokem je přidání proměnných do prostředí. Toto lze provést pomocí speciální UI záložky *Config Vars* servisu *Heroku* (záložka nastavení).

Obrázek 19: Uživatelské rozhraní servisu Heroku (sekce settings)



Zdroj: [15]

Po provedení všech výše uvedených kroků máme úspěšně nasazenou aplikaci do testovacího provozu.

Je nutno zmínit, že pro zajištění vysoké míry spolehlivosti běhu aplikace na produkci je třeba využít placený plán, který poskytne možnosti replikování, monitorování a spravování aplikace.

6 Další rozvoj aplikace

Architektura aplikace Bin Finder byla navržena tak, aby byla nejen snadno portovatelná, ale aby se dala použít i pro další města České republiky. Hlavní podmínkou pro rozšíření této webové aplikace na další města je existence OpenData zdrojů, které by obsahovaly data o poloze stanovišť pro tříděný komunální odpad. Pro zjištění náročnosti škálování byl proveden pokus o přidání dalšího města, který se skládal z několika standardních etap:

1) Vyhledávání a analýza OpenData zdrojů
2) Procesování a agregace dat
3) Umístění agregovaných dat do databáze
4) Ověření správného zobrazování dat

Během první etapy byl vyhledán OpenData zdroj *„Separovaný odpad“*[30] pro město Ostravu, který obsahoval data o poloze kontejnerů. Tento zdroj však obsahoval zcela omezený počet atributů pro zobrazení podrobných informací o kontejneru (pouze typ odpadu). Analýza datových zdrojů ukázala, že tento zdroj má odlišný formát datových záznamů. Pro standardizaci dat byl vytvořen skript, který by prováděl transformaci těchto zdrojů do GeoJSON standardu s předem definovaným seznamem vlastností, které tato aplikace využívá jako interní datový standard.

Spouštění tohoto skriptu se provádí pomocí terminálového příkazu:

```
yarn process:containers:ostrava
```

Kvůli neexistenci OpenData zdrojů, které by obsahovaly informaci o stanovištích na tříděný odpad, bylo třeba provést grupování kontejneru se stejnými souřadnicemi do stanovišť. Pro tuto transformaci byl vytvořen další skript, který provádí dané grupování.

Spouštění skriptů se provádí pomocí terminálového příkazu:

```
yarn process:stations:ostrava
```

Další etapou je umístění agregovaných dat do databáze aplikací. Odeslání dat se provádí pomocí UI desktop klientu MongoDB Altas (Příloha B).

Po úspěšném odeslání dat do databází lze buď přes UI aplikace ověřit zobrazení dat nebo pokud je aplikace zprovozněna pouze lokálně lze využít UI GraphiQL. Po absolvování těchto etap lze proces integraci města Ostravy do aplikací *Bin Finder* považovat za dokončený.

Nicméně aplikaci *Bin Finder* lze i dál vylepšovat, především jde o přidání dalších typů separovaného odpadu jako baterie a akumulátory, žárovky, nebezpečné odpady a přidání možností zobrazení sběrných dvorů na mapě.

Jako další krok na vylepšení aplikace je implementace portálů pro administraci, která by umožnila obnovení dat z OpenData zdrojů a zobrazení zpětných vazeb uživatelů pomocí UI.

Díky zvolené GraphQL architektuře pro pohodlnější využití aplikace na mobilních zařízeních lze také provést portaci aplikace i pro mobilní platformy *iOS* a *Android* beze změn serverní části aplikace.

Aplikace se plánuje udržovat v režimu testovacího provozu a v budoucnu se plánuje předání zdrojového kódu aplikací statutárnímu městu Praha pro nasazení do produkčního provozu.

7 Závěr

Cílem tohoto projektu bylo vytvořit webovou aplikaci pro vyhledávání volně přístupných kontejnerů pro tříděný komunální odpad, která by dokázala poskytnout uživateli informace o umístění kontejnerů na základě uživatelem předem nastavených voleb. Po provedení analýzy podobných softwarových řešení byl použit logický rámec, díky kterému byly stanoveny základní body pro realizaci projektu. Dále byly definované funkční a nefunkční požadavky projektu, které vyplynuly z analýzy podobných projektů.

Na základě čilé práce a analýzy funkčních požadavků bylo definováno několik základních uživatelských scénářů. Po definování jednotlivých kroků každého ze scénářů bylo zřejmě možno začít návrh designu aplikace. Pro pohodlné použití aplikace jak na desktopu, tak i na mobilním přístroji bylo rozhodnuto vytvořit responzivní design, který byl ve výsledku navrhnut pro základní form faktory: mobil, tablet a desktop.

V dalším kroku bylo rozhodnuto provést výběr programovacího jazyka a výběr základních knihoven projektu. Na základě osobních zkušeností a specifik projektů byly zvoleny ty nástroje, které splňují požadavky projektu. Následně bylo provedeno vyhledávání vhodných datových zdrojů. Na základě analýzy původu, podoby a specifik datových zdrojů aplikace bylo provedeno rozhodnutí o použití vhodného databázového systému. Dále následoval návrh architektury pro klientskou a serverní část a následná implementace navrhnutých řešení. V dalším kroku probíhalo psaní unit a snapshot testů a manuální testování aplikace. Pak bylo rozhodnuto provést nasazení aplikace do testovacího provozu.

V rámci této práce byl proveden úspěšný pokus o rozšíření aplikace i pro další město. Byl vyhledán a agregován další OpenData zdroj, který umožnil zobrazení nádob na tříděný odpad i pro město Ostravu.

Zásadním problémem z hlediska rozšíření aplikace pro další města je standardizace OpenData sad. Každé město rozhoduje o podobě, naplněnosti a formátu dat samostatně a nezávisle, proto není možné proces přidávání nových měst automatizovat. Dalším problémem je různorodost předpisů pro každé město. Města napříč celou ČR mají zcela odlišné podmínky sortování určitých typů odpadů (nápojové kartony, kov, barevné sklo), které jsou závislé především na dohodě s místními podniky, které zajišťují odvoz a další zpracování separovaného odpadu.

Jak je vidět z metrik volání *Google Maps JavaScript API* portálu *Google Cloud Platform*, během prvního týdne přes aplikaci bylo vyhledáno více než 70 stanovišť na tříděný komunální odpad.

Během testovacího provozu bylo zjištěno, že jedna třetina uživatelů chce využívat aplikaci v anglickém jazyce, proto bylo rozhodnuto přidat možnost lokalizace a angličtiny jako jednoho z možných jazyků rozhraní.

Pro pohodlnější zobrazení výsledků z vyhledávání bylo rozhodnuto přidat možnost skrývat hlavní panel aplikací pro zařízení s malým displejem.

8 Zdrojový kód projektu

Plná verze zdrojového kódu aplikace je zpřístupněna na servisu *Bitbucket* pro účet:

Adresa servisu: https://bitbucket.org

Username: jcu.bin.finder@gmail.com

Password: Prf.20$fZ

Domena aplikace *„Bin Finder“*: https://bin-finder.netlify.app

9 Slovník pojmů

SPA (Single Page Application) – webová aplikace, jejíž obsah je tvořen jedinou HTML stránkou a dynamicky se generuje za běhu.

UI (User Interface) – prostředek, kterým osoba ovládá softwarovou aplikaci nebo hardwarové zařízení. [20]

UX (User Experience) – zkušenost někoho s používáním produktu, systému nebo služby. [21]

OpenData – obsah, který může kdokoli libovolně sdílet za jakýmkoli účelem. [19]

API (Application Programming Interface) – sada příkazů, funkcí, protokolů a objektů, které programátoři mohou použít k vytvoření softwaru nebo k interakci s externím systémem. [22]

KANBAN – systém komunikace mezi pracovníky během výrobního procesu. [23]

GraphQL – open-source datový dotazovací a manipulační jazyk pro API. [24]

CRA (Create React App) – speciální CLI nástroj pro vytvoření ReactJS aplikací.

CLI (Command Line Interface) – je textové rozhraní používané pro zadávání příkazů. [25]

CI/CD (Continuous integration / Continuous delivery) – metoda častého doručování aplikací zákazníkům zavedením automatizace do fází vývoje aplikací. [26]

CRUD (Create, Read, Update, Delete) – operace s daty

DB (Database) – databázový systém

JSON (JavaScript Object Notation) – textový formát pro výměnu dat

CSS (Cascading Style Sheets) – tabulky kaskádových stylů se používají k formátování rozložení webových stránek. [27]

NPM – největší softwarový registr na světě. [28]

NoSQL (Not only SQL) – je přístup k návrhu databáze, který poskytuje flexibilní schémata pro ukládání a načítání dat nad rámec tradičních tabulkových struktur nalezených v relačních databázích. [29]

10 Seznam literatury

[1] *Mapa tříděného odpadu* [online]. 2019 [cit. 2020-01-01]. Dostupné z: https://ksnko.praha.eu/map-separated/

[2] *Sběrná místa separovaného odpadu* [online]. 2020 [cit. 2020-01-01]. Dostupné z: *http://nadoby.c-budejovice.cdsw.cz/*]

[3] *Mapový portál (Odpadové hospodářství)* [online]. 2020 [cit. 2020-01-01]. Dostupné z: *https://marushkapub.liberec.cz/*

[4] *Angular vs React vs Vue: Which is the Best Choice for 2019?* [online]. 2020 [cit. 2020-02-10]. Dostupné z: https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847

[5] *What is Apollo Server and what does it do* [online]. 2020 [cit. 2020-02-12]. Dostupné z: https://www.apollographql.com/docs/apollo-server/

[6] *Promise based HTTP client for the browser and Node.js* [online]. 2020 [cit. 2020-02-13]. Dostupné z: https://www.npmjs.com/package/axios

[7] *Request - Simplified HTTP client* [online]. 2020 [cit. 2020-02-14]. Dostupné z: https://www.npmjs.com/package/request

[8] *React Apollo* [online]. 2020 [cit. 2020-02-15]. Dostupné z: https://www.npmjs.com/package/react-apollo

[9] *React Apollo. Local state management* [online]. 2020 [cit. 2020-02-20]. Dostupné z: *https://www.apollographql.com/docs/react/data/local-state/*

[10] *Google Map React* [online]. 2020 [cit. 2019-02-20]. Dostupné z: https://www.npmjs.com/package/google-map-react

[11] *React Google Maps* [online]. 2020 [cit. 2019-03-10]. Dostupné z: https://www.npmjs.com/package/react-google-maps

[12] *Styled Components* [online]. 2020 [cit. 2019-03-12]. Dostupné z: https://www.npmjs.com/package/styled-components

[13] *Heroku CLI tool* [online] 2020 [cit. 2020-05-11]. Dostupné z: https://devcenter.heroku.com/articles/heroku-cli#download-and-install

[14] *Netlify hosting service* [online] 2020 [cit. 2020-05-22]. Dostupné z: https://app.netlify.com/

[15] *Heroku hosting service* [online] 2020 [cit. 2020-06-17]. Dostupné z: https://dashboard.heroku.com/apps

[16] *Heroku settings dashboard* [online] 2020 [cit. 2020-06-20]. Dostupné z: https://dashboard.heroku.com/apps/bin-finder-server/settings

[17] *OpenData sada „Stanoviště tříděného odpadu – položky“* [online] 2020 [cit. 2020-01-24]. Dostupné z: [https://www.geoportalpraha.cz/cs/data/metadata/8B2B3E58-1B3F-4870-B205-82453E90A2F8]

[18] *OpenData sada „Stanoviště tříděného odpadu“* [online] 2020 [cit. 2021-01-24]. Dostupné z: https://www.geoportalpraha.cz/cs/data/metadata/8726EF0E-0834-463B-9E5F-FE09E62D73FB

[19] *Pojem OpenData* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://www.yourdictionary.com/open-data

[20] *Pojem UI* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://techterms.com/definition/user_interface

[21] *Pojem UX* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://dictionary.cambridge.org/dictionary/english/user-experience

[22] *Pojem API* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://techterms.com/definition/api

[23] *Pojem KANBAN* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://dictionary.cambridge.org/dictionary/english/kanban?q=KANBAN

[24] *Pojem GraphQL* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://www.definitions.net/definition/GRAPHQL

[25] *Pojem CLI* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://techterms.com/definition/command_line_interface

[26] *Pojem CI/CD* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[27] *Pojem CSS* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://techterms.com/definition/css

[28] *Pojem NPM* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://www.w3schools.com/whatis/whatis_npm.asp

[29] *Pojem NoSQL* [online] 2020 [cit. 2021-01-15]. Dostupné z: https://www.ibm.com/cloud/learn/nosql-databases

[30] *OpenData sada „Separovaný odpad“* [online] 2021 [cit. 2021-01-24]. Dostupné z: [https://data.gov.cz/datov%C3%A1-sada?iri=https%3A%2F%2Fdata.gov.cz%2Fzdroj%2Fdatov%C3%A9-sady%2F00845451%2F37085087]

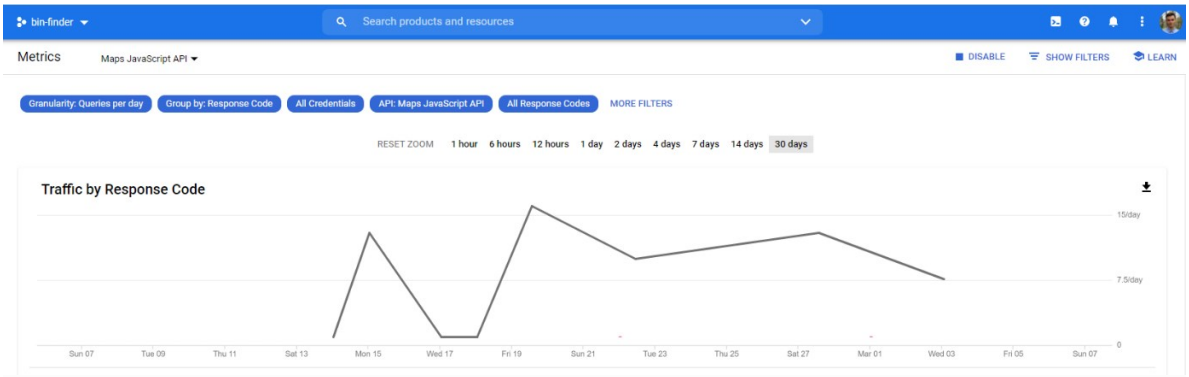
11 Seznam obrázků

12 Seznam tabulek

Přílohy

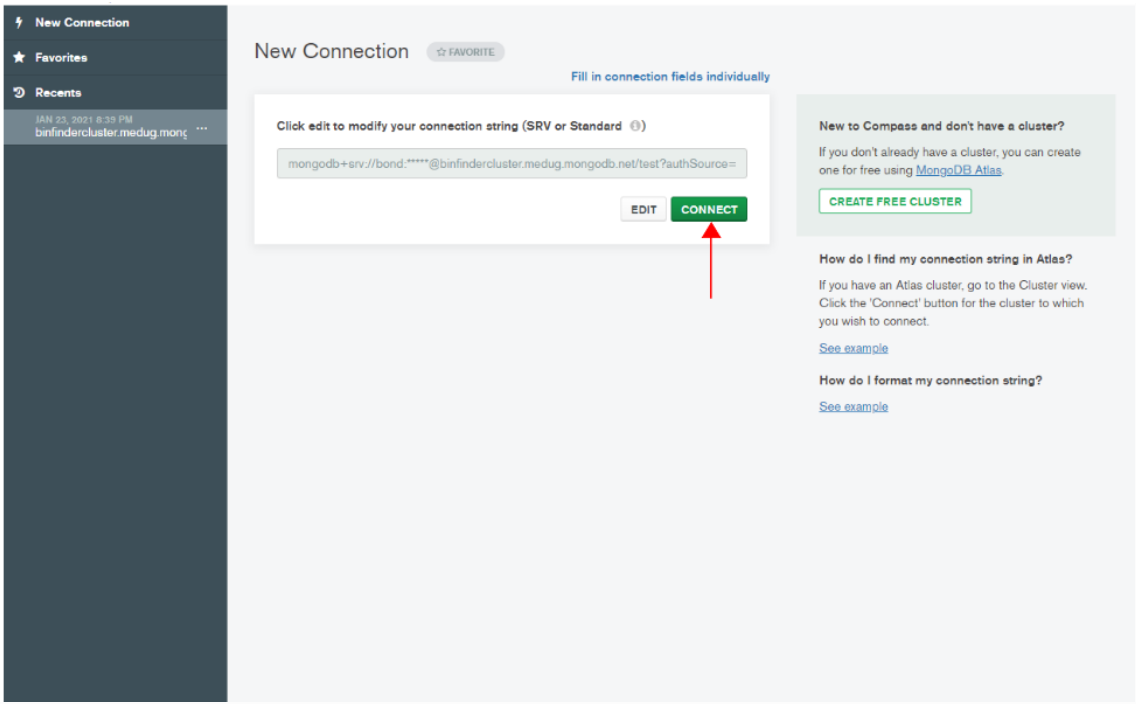
Příloha A

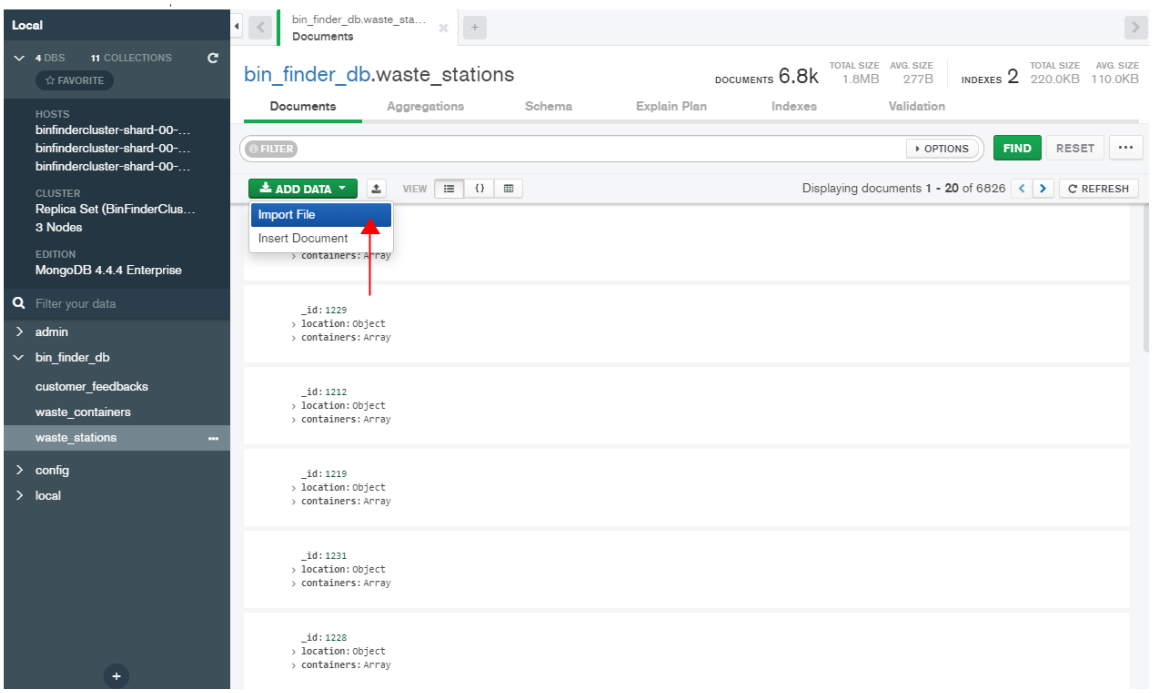
Metrika využití Google Maps JavaScript API

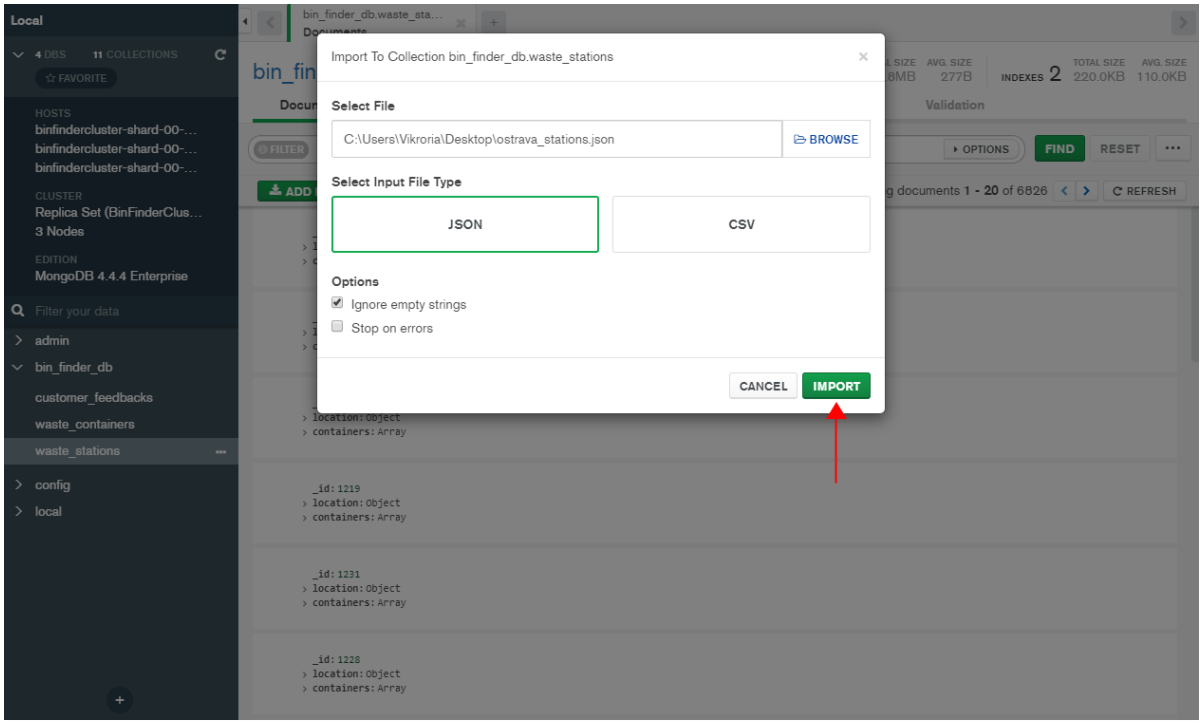


Příloha B

Proces přidání dat do MongoDB pomoci MongoDB Atlas






Local
4 DBS
11 COLLECTIONS
FAVORITE
HOSTS
binfindercluster-shard-00-...
binfindercluster-shard-00-...
binfindercluster-shard-00-...
CLUSTER
Replica Set (BinFinderClus...
3 Nodes
EDITION
MongoDB 4.4.4 Enterprise
Filter your data
admin
bin_finder_db
customer_feedbacks
waste_containers
waste_stations
config
local
Import To Collection bin_finder_db.waste_stations
Select File
C:\Users\Vikroria\Desktop\ostrava_stations.json
BROWSE
Select Input File Type
JSON
CSV
Options
Ignore empty strings
Stop on errors
CANCEL
IMPORT
FIND
RESET
REFRESH