

Střední průmyslová škola elektrotechnická
a Vyšší odborná škola Pardubice

**STŘEDNÍ PRŮMYSLOVÁ ŠKOLA
ELEKTROTECHNICKÁ**

MATURITNÍ PRÁCE – WEBOVÉ STRÁNKY

SYSTÉM PRO HLEDÁNÍ KONTAKTNÍCH ÚDAJŮ

„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.

Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejích ukázek pro výuku.“

V Pardubicích dne

.....

podpis



ZADÁNÍ MATURITNÍ PRÁCE

Maturitní zkouška – profilová část – Maturitní projekt

Obor:	18-20-M/01 Informační technologie	Školní rok:	2023/2024
Jméno a příjmení žáka:	Filip Radosta	Třída:	4.D
Téma maturitní práce:	Systém pro hledání kontaktních údajů		
Vedoucí maturitní práce:	Mgr. Čestmír Bárta		
Pracoviště vedoucího:	SPŠE a VOŠ Pardubice		

Kategorie maturitní práce: Web

Téma maturitní práce (popis):

Vytvořte systém pro hledání kontaktních údajů. Na webové stránce uživatel zadá, co hledá a lokaci, například kavárny v Pardubicích. Aplikace vrátí z údajů uložených v databázi (data z OpenStreetMap) příslušná data (tzn. název, adresa, email, telefon, web). Data budou zobrazena v tabulce, ve které půjdou data filtrovat, nebo bude možné data stáhnout. Administrátor aplikace bude mít možnost databázi aktualizovat novými daty a bude mít možnost zobrazit a upravovat data uživatelů webu.

Hlavní body – specifikace maturitní práce:

- 1) Zpracování dotazu a zobrazení hledaných dat v tabulce.
- 2) Filtrování dat v tabulce a možnost stáhnout data z tabulky.
- 3) Možnost nahrát nová data do databáze.
- 4) Zabezpečení pomocí captcha.
- 5) Možnost zobrazit a manipulovat s uživateli webu.

Způsob zpracování maturitní práce:

Maturitní práce musí být zpracována v souladu se zákonem č. 121/2000 Sb., autorský zákon.

Maturitní práce je tvořena praktickou částí (viz téma a specifikace maturitní práce výše) a písemnou prací.

Maturitní práce je realizována žákem převážně v rámci výuky ve 4. ročníku. Lze pokračovat na projektu z nižších ročníků.

Podrobné pokyny ke zpracování maturitní práce příslušné kategorie jsou uvedeny v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Zpracování písemné práce musí odpovídat požadavkům uvedeným v dokumentu *MP_Formalni_stranka_dokumentace*. Písemná práce musí být rovněž zpracována v souladu s normou pro úpravu písemností [ČSN 01 6910 (2014)], s citační normou [ČSN ISO 690], se základními typografickými pravidly, pravidly sazby, gramatickými pravidly a pravidly českého pravopisu.

Pokyny k rozsahu a obsahu maturitní práce:

Rozsah a obsah praktické části maturitní práce je určen tématem a specifikací maturitní práce, rozsah písemné části maturitní práce je minimálně 15 normostran vlastního textu. Do uvedeného rozsahu se nezapočítávají úvodní listy (titulní list, prohlášení, zadání, anotace, obsah...), závěrečné listy (seznam literatury...) a přílohy. Podrobné pokyny k rozsahu a obsahu maturitní práce příslušné kategorie jsou uvedeny v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Kritéria hodnocení maturitní práce a její obhajoby:

Maturitní práce a její obhajoba u maturitní zkoušky je hodnocena bodově. Celkový dosažený počet bodů je součtem bodů přidělených vedoucím práce a oponentem (maximální dosažitelný počet je 100 bodů). Výsledná známka u maturitní zkoušky je stanovena přepočtem celkového počtu bodů na známku pomocí tabulky uvedené níže.

V případě nesplnění tématu maturitní práce a v případě plagiátorství bude práce hodnocena stupněm „nedostatečný“.

Tabulka bodového vyjádření hlavních kritérií a obhajoby:

- praktická část – zpracování tématu maturitní práce[max. 25 bodů]
- písemná část – zpracování, formální a obsahová stránka[max. 15 bodů]
- obhajoba maturitní práce před zkušební komisí[max. 10 bodů]

Podrobná kritéria pro hodnocení maturitní práce příslušné kategorie jsou uvedena v dokumentu *MP_Kriteria_hodnoceni*.

Tabulka přepočtu celkového počtu bodů na známku:

[0 .. 60 bodů]	[5]
[61 .. 70 bodů]	[4]
[71 .. 80 bodů]	[3]
[81 .. 90 bodů]	[2]
[91 .. 100 bodů]	[1]


Požadavek na počet vyhotovení maturitní práce a její odevzdání:

Kompletní maturitní práce (praktická i písemná) se odevzdává ve stanoveném termínu vedoucímu maturitní práce. Písemná práce se odevzdává v jednom tištěném vyhotovení obsahující podepsaný přenosný nosič CD/DVD/SD s písemnou prací a sadou kompletních dat v elektronické podobě dle pokynů v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Termín odevzdání maturitní práce: 5. dubna 2024

Délka obhajoby maturitní práce před zkušební maturitní komisí: 15 minut

Pardubice 23. října 2023


Mgr. Petr Mikuláš, ředitel školy

Anotace

Webová aplikace pro hledání kontaktních údajů, určena například pro začínající podnikatele hledající nové zákazníky mezi již existujícími podniky. Umožní zobrazit a filtrovat data v tabulce s možností ji stáhnout. Umožňuje správu dat a uživatelů.

Klíčová slova: kontakty, vyhledávání, podnik

Annotation

Web application for finding contact details, for example for start-ups looking for new customers among existing businesses. Allows you to view and filter data in a downloadable spreadsheet. Allows data and user management.

Keywords: contacts, search, enterprise

Obsah

ÚVOD...	8
1 ANALÝZA OBDOBNÝCH WEBOVÝCH STRÁNEK	9
1.1 GRABCONTACTS	9
1.1.1 Kladné stránky	10
1.1.2 Záporné stránky	10
1.2 SCRAP.IO	11
1.2.1 Kladné stránky	11
1.2.2 Záporné stránky	12
1.3 EMAIL SCRAPER	12
1.3.1 Kladné stránky	13
1.3.2 Záporné stránky	13
2 NÁVRH PROJEKTU	14
2.1 CÍLOVÉ SKUPINY	14
2.2 ADMINISTRACE WEBU	14
2.2.1 Use case diagram	17
2.3 DATABÁZE	17
2.4 DESIGN A RESPONZIVITA	19
3 POPIS PROJEKTU	21
3.1 FRONTEND	21
3.1.1 HTML	21
3.1.2 CSS.....	22
3.1.3 JavaScript	23
3.2 BACKEND	28
3.2.1 PHP.....	28
ZÁVĚR	37
SEZNAM PŘÍSTUPOVÝCH ÚDAJŮ	38
SEZNAM POUŽITÉ LITERATURY A ZDROJŮ OBRÁZKŮ	39

Úvod

Jako maturitní projekt jsem se rozhodl vytvořit jednoduchou webovou aplikaci, která bude mít za cíl snadné získávání kontaktních údajů. Dovolte mi krátce vysvětlit, proč jsem si vybral právě toto téma.

Už mě napadlo několik podnikatelských nápadů, které jsem samozřejmě nikdy nerealizoval, nicméně většina z nich pokulhávala na problému, jak získat zákazníky pro daný produkt, ať už hmotný nebo nehmotný. Marketing ve formě “kobercového bombardování” reklamou je drahý, a tak jsem se začal zabývat “microtargetingem” reklamy.

Napadlo mě vytvořit systém, který kontaktuje nebo zobrazí reklamu přesně takovému zákazníkovi, kterého potřebuji. Uvedu to na příkladu. Řekněme, že jsem založil firmu na pražení kávy a nemám ji komu prodat. Pomocí takového systému ale mohu automaticky najít a kontaktovat všechny kavárny a obchody široko daleko a nabídnout jim produkt.

Logickým prvním krokem k vytvoření projektu, který jsem tu nastínil, je získání velkého množství dostatečně kvalitních kontaktních údajů a systém, který je zpracuje k dalšímu použití, a právě to je cílem tohoto maturitního projektu.

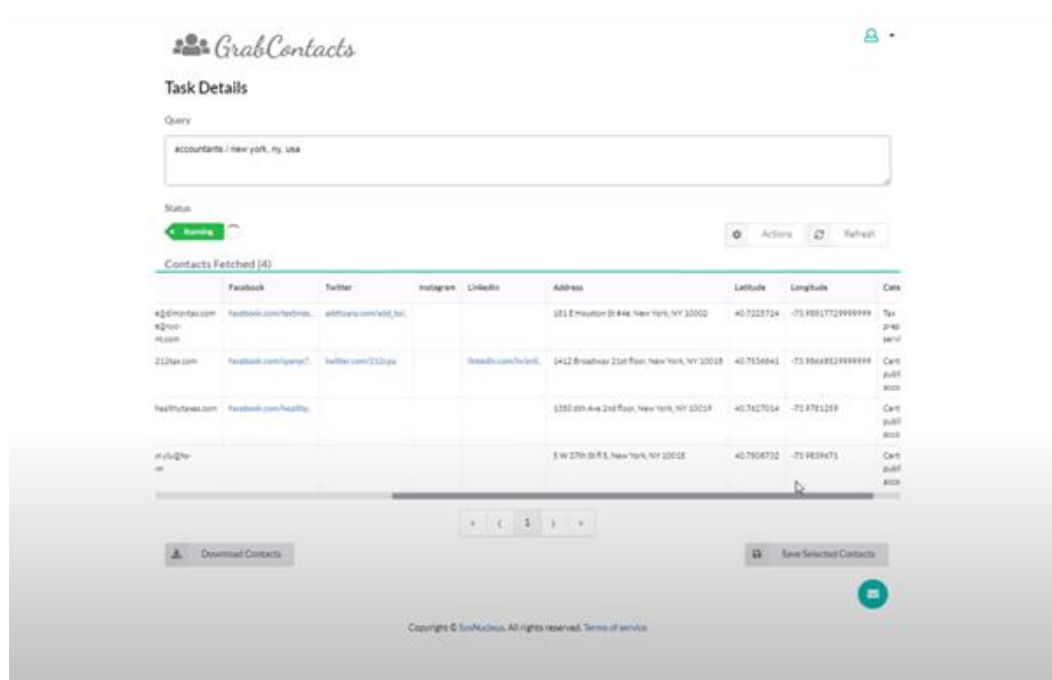
Podobných aplikací, které slouží ke hledání kontaktních údajů není mnoho, ale přece jich na internetu lze několik najít. Jejich nevýhodou je ale vysoká cena nebo nízký počet výsledků v České republice, a proto jsem se rozhodl vytvořit vlastní webovou aplikaci.

1 Analýza obdobných webových stránek

Analýza je důležitým prvkem při vývoji nové aplikace, a to zejména proto, abychom se vyvarovali slabým stránkám konkurence a dokázali nabídnout něco nového.

1.1 GrabContacts

Adresa: grabcontacts.com



Obrázek 1: GrabContacts (zdroj: grabcontacts.com)

Nástroj Grabcontacts vykonává svoji práci velmi obstojně, díky velikosti google maps, ze kterých web čerpá, najde velký počet výsledků. Někoho by ale mohl odradit méně intuitivní design nebo ne úplně rychlé hledání oproti konkurenci.

1.1.1 Kladné stránky

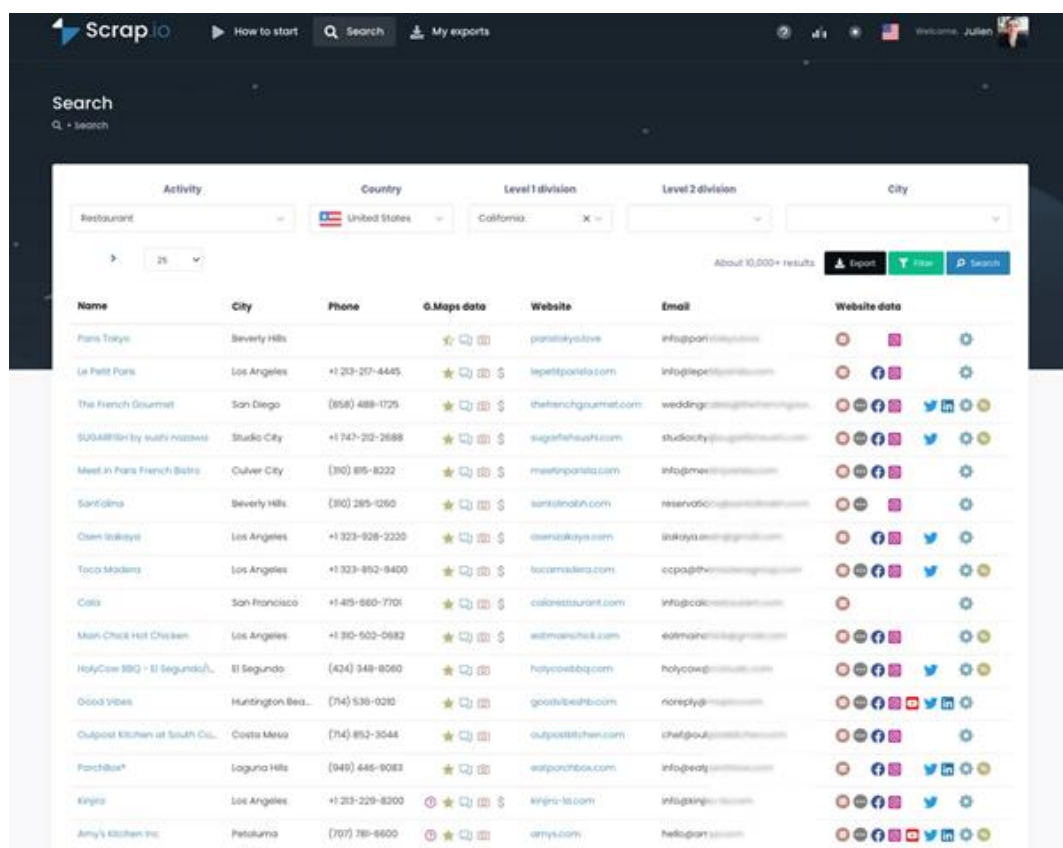
- Data z map i webových stránek dostupná v tabulce
- Hledání jednoduše podle názvu města
- Poskytují dostatek kvalitních dat i ve městech jako Pardubice
- Možnost výsledek stáhnout nebo uložit

1.1.2 Záporné stránky

- Pokud chci vidět, kde se hledaný podnik nachází, musím ručně kopírovat souřadnice
- Z velké části placená služba
- Nelze nijak filtrovat

1.2 Scrap.io

Adresa: [Scrap.io](https://scrap.io)



Obrázek 2: Scrap.io (zdroj: alternativeto.net)

Nástroj Scrap.io se pyšní nejen moderním designem a jednoduchým ovládáním, ale i například pokročilou možností filtrování. Oproti konkurenci ale ztrácí účinnost v ČR, a navíc je to celkem drahá služba.

1.2.1 Kladné stránky

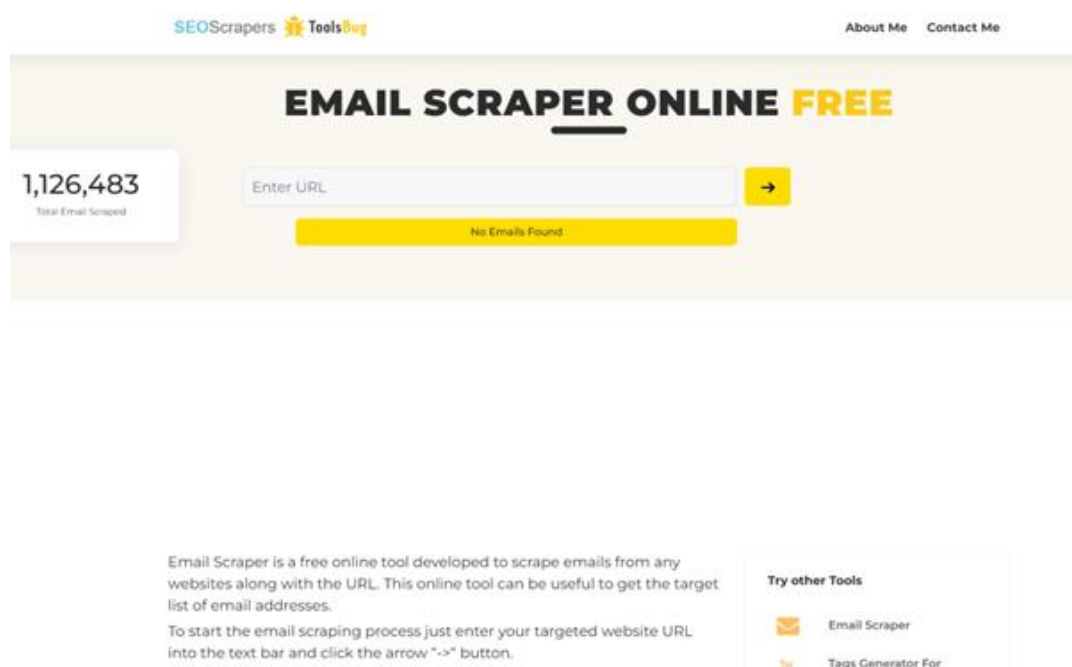
- Data dostupná v přehledné tabulce, intuitivní design
- Prohledává mapy i webové stránky
- Výsledek vrací okamžitě
- Možnost stáhnout, filtrovat, hledat v tabulce

1.2.2 Záporné stránky

- Malý počet výsledků minimálně v ČR
- Placená služba
- Například Praha je rozdělena na části, nelze prohledat celou

1.3 Email Scraper

Adresa: [Email Scraper](#)



Obrázek 3: Email Scraper (zdroj: vlastní)

Email Scraper Online je nástroj pro sběr emailových adres z webových stránek. Nástroj nabízí jednoduchý design, kde uživatel zadá url adresu a web mu vrátí csv soubor ke stažení. Problémem je, že se občas zasekne, nebo nic nenajde a nezobrazuje výsledky, jen soubor, ve kterém jsou navíc duplicitní záznamy.

1.3.1 Kladné stránky

- Zcela zdarma
- Přehledný design
- Možnost výsledky stáhnout

1.3.2 Záporné stránky

- Nehledá url adresy a kontakty v nich, pouze kontakty v uživateli zadané url adresy
- Hledá pouze emaily, navíc jsou ve výsledku duplicitní
- Často žádné emaily nenajde

2 Návrh projektu

2.1 Cílové skupiny

Webové stránky mohou využít podnikatelé hledající zákazníky mezi firmami nebo kdokoliv, kdo pro zlato dnešní doby – data, v tomto případě kontakty, najde využití.

2.2 Administrace webu

Webová aplikace je spravována administrátory, což jsou registrovaní uživatelé webu, kteří mají roli „ADMIN“. Administrátorovi se po přihlášení zobrazí v horním panelu link na admin panel, kde se nachází správa dat a uživatelů. Ve správě dat lze nahrát zazipovaný JSON soubor striktně ve specifickém formátu. Toho lze dosáhnout například následujícím postupem: nejprve je nutné stáhnout osm soubor dostupný zde: <https://download.geofabrik.de/europe/czech-republic.html>. Ten je ale příliš velký, aby bylo možné ho nahrát na server, je nutné ho proto zpracovat na JSON s pouze užitečnými daty, který musí mít parametry stejné jako v tomto příkladu:

```
{  "amenity": "school",
  "email": "spse@spse.cz",
  "name": "SP\u00160E a VO\u00160",
  "phone": "+420 466 614 788",
  "website": "https://www.spse.cz/",
  "coordinates": {
    "lat": 50.0372915,
    "lon": 15.7794835
  },
  "address": {
    "houseNumber": "13",
    "street": "Karla IV.",
    "postcode": "53002",
    "city": "Pardubice",
    "country": "",
    "suburb": ""
  }
}
```

Takový JSON soubor lze získat například s použitím již zmíněného osm souboru a následujícího kódu na lokálním serveru:

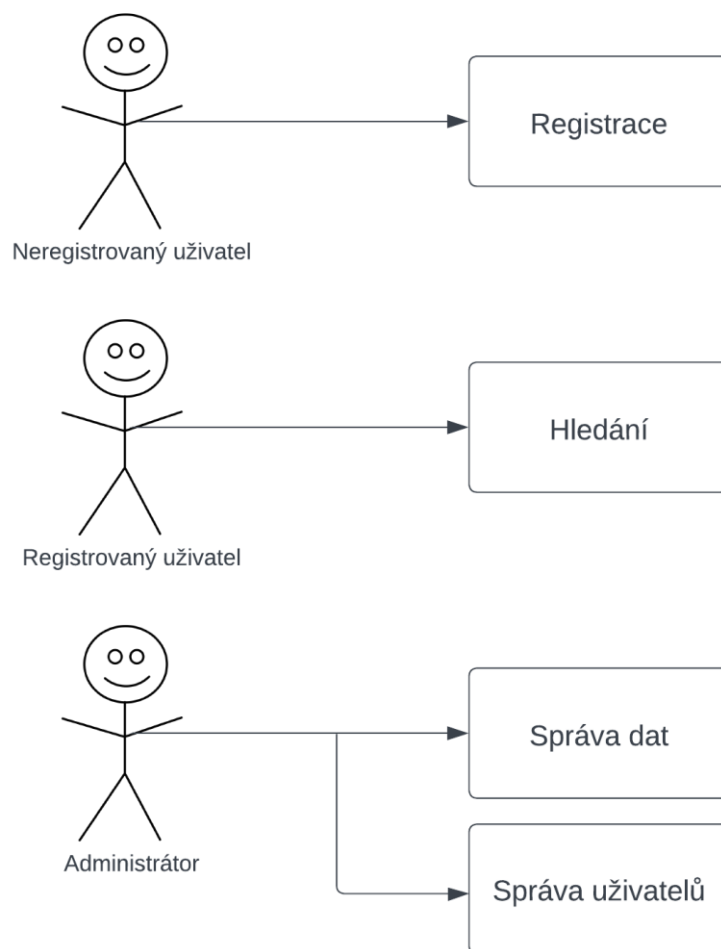
```
<?php $reader = new XMLReader(); $reader->open('czech-republic-latest.osm');
$dom = new DOMDocument; $businessesByAmenity = []; $allAmenities = []; while
($reader->read()) { if ($reader->nodeType == XMLReader::ELEMENT && $reader-
>name == 'node') { $node = $dom->importNode($reader->expand(), true);
$simpleXmlNode = simplexml_import_dom($node); $businessData = ['amenity' =>
'', 'email' => '', 'name' => '', 'phone' => '', 'website' => '',
'coordinates' => ['lat' => (float)$simpleXmlNode['lat'], 'lon' =>
(float)$simpleXmlNode['lon']], 'address' => ['houzenumber' => '', 'street'
=> '', 'postcode' => '', 'city' => '', 'country' => '', 'suburb' => '',],];
$amenity = null; foreach ($simpleXmlNode->tag as $tag) { $key =
(string)$tag['k']; $value = (string)$tag['v']; switch ($key) { case
'amenity': $amenity = $value; $businessData['amenity'] = $value; break; case
'email': $businessData['email'] = $value; break; case 'name':
$businessData['name'] = $value; break; case 'phone': $businessData['phone']
= $value; break; case 'website': $businessData['website'] = $value; break;
case 'addr:housenumber': $businessData['address']['houzenumber'] = $value;
break; case 'addr:street': $businessData['address']['street'] = $value;
break; case 'addr:postcode': $businessData['address']['postcode'] = $value;
break; case 'addr:city': $businessData['address']['city'] = $value; break;
case 'addr:country': $businessData['address']['country'] = $value; break;
case 'addr:suburb': $businessData['address']['suburb'] = $value; break; } }
if (!empty($businessData['name']) && !empty($amenity)) { if
(!empty($businessData['email']) || !empty($businessData['phone']) ||
!empty($businessData['website'])) { if
(!isset($businessesByAmenity[$amenity])) $businessesByAmenity[$amenity] =
[]; $businessesByAmenity[$amenity][] = $businessData; }
$allAmenities[$amenity] = true; } } } $reader->close(); $businessesJson =
json_encode($businessesByAmenity, JSON_PRETTY_PRINT);
file_put_contents('objekty.json', $businessesJson); $amenitiesJson =
json_encode(array_keys($allAmenities), JSON_PRETTY_PRINT);
file_put_contents('kategorie.json', $amenitiesJson); echo 'Objekty byly
extrahovány, roztrženy podle kategorie a uloženy do objekty.json.'; echo
'Unikátní kategorie byly extrahovány do kategorie.json.';
```

Následně je nutné JSON zazipovat, abychom na server nahrávali co nejmenší soubor, z důvodu limitů serveru. Tímto postupem je možné původní 15 GiB soubor zmenšit na zhruba 0.8 MiB, což je efektivní a zrychluje to práci. Nahrávat může jen jeden admin jeden soubor najednou, aby se předešlo problémům.

Ve správě uživatelů se nachází tabulka všech uživatelů, kde lze uživatele smazat nebo vybrat, v takovém případě se data vybraného uživatele objeví v upravovacím formuláři pod tabulkou, ve kterém může admin upravit jméno, mail a heslo uživatele. V případě že změní heslo, tak se heslo zahashuje a pošle do databáze, jinak zůstává stejné a znovu se zahashuje. Dále správa uživatelů nabízí i formulář na přidání uživatele. Po mazání, přidání nebo upravení se objeví upozornění s potvrzením operace.

2.2.1 Use case diagram

Na use case diagramu můžeme vidět chování webu z uživatelského pohledu, rozdělený na tři typy uživatelů stránky. K prohledávání dat mají přístup všichni uživatelé, na rozdíl od správy webu, ke které má přístup pouze administrátor.



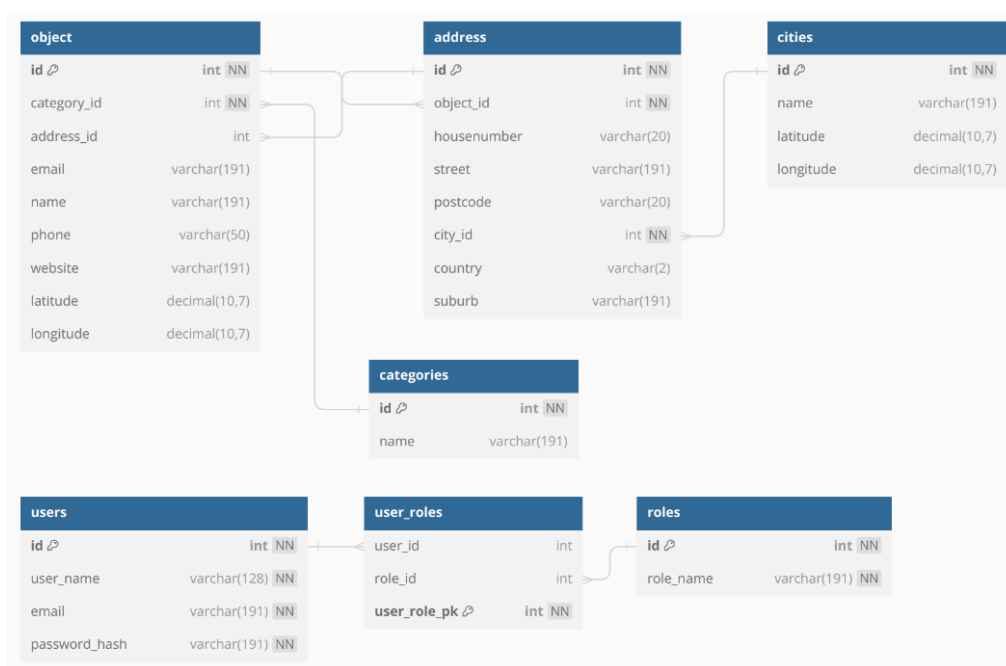
Obrázek 4: Use case diagram (zdroj: vlastní)

2.3 Databáze

Každý záznam v nahraném JSONu se v databázi rozdělí na záznamy v tabulce object. V té můžeme také najít id kategorie a adresy, které se ukládají

v samostatných tabulkách. Dále jsou v databázi tabulky roles s rolemi a users s uživateli, a protože uživatel může mít více rolí a naopak, tak se mezi těmito tabulkami nachází ještě tabulka user_roles. V databázi ještě můžeme najít zálohové tabulky, které jsou součástí zálohovacího mechanismu při nahrávání nových dat.

Kromě tabulek zabývajících se uživateli a rolemi, které běží na InnoDB storage engine, zbytek databáze používá MyISAM, protože povaha využití těchto tabulek nevyžaduje referenční integritu, kterou zmíněný engine nenabízí. Admin nemá možnost mazání jednotlivých řádků daných tabulek, takže se nemůže porušit integrita dat. Výhodou je pak rychlejší práce s daty, což je v databázi s desítkami tisíc řádků rozhodující. Znatelné je to především u čtení, a protože jsou dané tabulky vytíženy právě primárně čtením, tak je tato vlastnost klíčová.



Obrázek 5: Schéma databáze (zdroj: vlastní)

2.4 Design a responzivita

Moderní web design si zakládá na čistotě a přehlednosti, a proto jsem zvolil plochý jednostránkový design. Pozadí stránky tvoří neutrální bílá a případně světlé designové prvky. Na tlačítka jsem použil výraznou červenou barvu, aby přilákala uživatelské oko. Logo a obrázky použité jako pozadí jsem vygeneroval modelem strojového učení DALL-E 3, aby stránka působila moderním a neobvyklým dojmem. Pozadí datové tabulky a přihlašovací stránky tvoří geometrické tvary v barevném stylu stránky. Dalším charakteristickým prvkem designu jsou zaoblené rohy elementů.



Obrázek 6: Přihlašovací stránka (zdroj: vlastní)

Datová tabulka využívá průhlednosti jako moderní designový prvek. Všechny stránky webu nesou stejné nebo podobné designové prvky a jsou responzivní, a to primárně kolem hodnot 680, 920 a 1400 pixelů. V mobilní verzi stránky se navigační menu schová do hamburger menu.



Obrázek 7: Ukázka responzivity (zdroj: websiteplanet.com)

3 Popis projektu

Webová aplikaci funguje na principu klient server. Tyto dvě oddělené vrstvy, PHP na serveru a JavaScript na klientovi, mezi sebou komunikují pomocí AJAX (Asynchronous JavaScript and XML), což umožňuje výměnu dat na pozadí bez nutnosti obnovovat celou stránku. Na straně klienta je použit JavaScript k odesílání požadavků na server, kde PHP skripty zpracovávají tato data a vracejí odpovědi zpět klientovi. Výhodou je také to, že HTML, JavaScript i PHP jsou oddělené, každý ve svém souboru pro lepší čitelnost.

Projekt je zabezpečený proti SQL injekcím, díky použití připravených dotazů a XSS útokům, díky escapování dotazů.

3.1 Frontend

Web obsahuje hlavní, uživatelskou, administrační, přihlašovací, registrační a odhlašovací stránku. Každá z nich má svůj HTML soubor. HTML soubor navigačního panelu je načítán JavaScriptem na horní stranu všech stránek. Tabulka s daty je vkládána do hlavní a uživatelské stránky pomocí iframe z důvodu stylování, které rozeberu v části o CSS. Web je responzivní pomocí Media Queries na desktop, tablet a telefon.

3.1.1 HTML

Tělo hlavní stránky je rozděleno na obalující prvek, náhledový obrázek a datové tabulky, které kvůli jejich plné šířce nejsou v obalovém prvku, ve kterém se pak nachází místo na načtení navigačního panelu, několik sekcí a patičky, na které je stránka rozdělená. Další stránky jsou členěné na podobném principu. Díky Sass nestování a externí CSS knihovně jsou HTML soubory nezahlcené přebytečnými identifikacemi id.

3.1.1.1 Obecně o HTML

Hypertext Markup Language, nebo zkráceně HTML je v informatice název pro značkovací jazyk, který se používá na tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. HTML je hlavním jazykem pro tvorbu stránek v systému World Wide Web, který umožňuje sdílení dokumentů na internetu. Jazyk vychází z staršího univerzálního značkovacího jazyka SGML. Design stránky se pak tvoří za pomoci kaskádových stylů CSS.

3.1.2 CSS

Kaskádové styly CSS slouží k formátování HTML stránek pomocí nastavování stylů nadpisů, odstavců, odkazů, nebo obrázků i celých částí stránky. CSS se vyvíjí průběžně s HTML, v současné době se používá standard CSS3, který podporuje řadu nových vlastností.

3.1.2.1 Sass

Technologii Sass jsem použil kvůli nestování CSS kódu, což zjednodušuje psaní a zlepšuje čitelnost. Pomocí pluginu na kompilování Sass kódu jsem zkompiloval Sass kód na klasický CSS soubor. Nesting, o kterém jsem psal, vypadá následovně:

```
.logo {  
  display: flex;  
  align-items: center;  
  
  img {  
    width: 6em;  
  }  
  
  a {  
    color: black;  
    font-weight: 600;  
  }  
}
```

3.1.2.2 Water.css

Jako základní kámen stylů jsem použil CSS knihovnu water.css, která usnadnila stylování především formulářů. Tuto knihovnu využívá každá stránka webové aplikace a je připojena v hlavičce dokumentu.

Vzhledem k negativnímu vlivu water.css na styly datové tabulky, kde styly knihovny ovlivňovaly nebo přepisovaly mé vlastní styly, jsem byl nucen celou tabulku a její styly schovat do iframe elementu, aby water.css ovlivňovala pouze zbytek stránky. V HTML iframe vypadá následovně:

```
<iframe src="datatable.html" width="100%" height="100vh"
id="iframe"></iframe>
```

Na klientské části uživatelské stránky se pak přistupuje k iframe takto:

```
let iframeDocument = document.querySelector('#iframe').contentDocument;
```

3.1.3 JavaScript

Jazyk Javascript byl vytvořený jako prostředek pro přidávání interaktivity do webových stránek. Postupným vývojem se ale z jednoduchého skriptovacího jazyka pro webové stránky stal nezbytnou součástí moderního webového vývoje a rozšířil se i do oblastí mimo webové prohlížeče. Díky jeho univerzálnosti a široké podpoře je jeden z nejpoužívanějších jazyků.

V projektu jsem použil JavaScriptovou knihovnu JQuery, která dává důraz na propojení jazyka s HTML a usnadňuje tak práci s elementy DOM.

3.1.3.1 Role a neaktivita na klientovi

Veškeré validace zadaných údajů uživatelem, kontrola jeho role nebo kontrola neaktivity probíhá jak na serverové části, kvůli bezpečnosti, tak na

klientské části, kvůli odlehčení práce serveru. Neaktivita se měří jako 15 minut od aktivity na myši nebo klávesnici:

```
document.addEventListener('mousemove', resetLogoutTimer);
document.addEventListener('keypress', resetLogoutTimer);
document.addEventListener('click', resetLogoutTimer);
```

Role se na klientské části kontroluje dotazem na server na začátku skriptu.

3.1.3.2 *JustValidate.js*

Validace například na registrační stránce je řešena pomocí JavaScriptové knihovny JustValidate, která poskytuje předdefinovaná pravidla pro validaci, která je pak jednodušší a čitelnější:

```
validation
  .addField('#name', [
    {
      rule: 'required'
    },
    {
      validator: (value) => {
        return value.length <= 12;
      },
      errorMessage: 'Jméno musí mít maximálně 12 znaků.'
    }
  ])
])
```

3.1.3.3 *Přihlášení a registrace na klientovi*

Klientská část přihlašovací stránky naslouchá přihlašovacímu formuláři a při odeslání dat uživatelem se zkontroluje, zda je vyplněná captcha, aby se zamezilo zbytečnému přehlcování serveru:

```
const captchaResponse = grecaptcha.getResponse();
if (!captchaResponse.length > 0) {
  document.querySelector('#errorMsg').innerText = 'Vyplňte captcha test.';
  document.querySelector('#errorMsg').style.display = 'block';
}
```


V případě vyplnění uživatelem se pošle POST dotaz na server a při úspěchu proběhne přihlášení, v případě neúspěchu se vypíše chybová hláška:

```
fetch('php/login.php', {
  method: 'POST',
  body: formData
})
.then(response => response.json())
.then(data => {
  if (data.success) {
    window.location.href = "user_page.html";
  } else {
    document.querySelector('#errorMsg').innerText = data.message;
  }
});
```

Klientská část registrační stránky funguje na stejném principu.

3.1.3.4 Administrátorská část na klientovi

Administrátorská část má na starosti hned několik úkolů, v první řadě posílá POST dotazem nahraný soubor na server. Druhým úkolem skriptu je nahrát z databáze všechny uživatele a přidat k nim tlačítko upravit a smazat. Při úpravě, mazání a přidávání uživatele slouží JavaScript jako prostředník při posílání dat z HTML do PHP. Jedná se zpravidla o uživatelské id, případně nová data při úpravě nebo přidání uživatele. Před odesláním dotazu na server se systém ptá na správnost údajů.

```
document.querySelectorAll('.deleteBtn').forEach(button => {
  button.addEventListener('click', function() {
    const userId = this.getAttribute('data-user-id');
    if (confirm('Opravdu chcete smazat tohoto uživatele?')) {
      fetch(`php/delete_user.php?user_id=${userId}`, {method: 'POST'})
        .then(response => response.json())
        .then(data => {data.success ? fetchUsers() : alert(
          'Chyba při mazání uživatele.')});
    }
    .catch(error => console.error('Error:', error));
  });
});
```

3.1.3.5 Uživatelská část na klientovi

Klientská část uživatelské stránky má za úkol především poslat POST dotazem na server formulářová data, která server zpracuje a pošle zpátky odpověď, v tomto případě pole, které je třeba rozdělit a vložit do tabulky na klientovi.

```
data.forEach(obj => {
  let row = tbody.insertRow();
  let address = `${obj.housenumber} ${obj.street}, ${obj.postcode},
${obj.city}, ${obj.country}, ${obj.suburb}`;
  [obj.name, obj.email, obj.phone, obj.website, obj.latitude,
obj.longitude, address].forEach(text => {
    let cell = row.insertCell();
    cell.textContent = text;
  });
});
iframeDocument.querySelector('table').appendChild(tbody);
```

Tím se vloží data do tabulky, která ale zatím není z hlediska klienta funkční. Její skript, který do ní vkládá posluchače událostí. Tento skript, protože je napojený na tabulku v iframe funguje odděleně, a tak je potřeba ho při každém vkládání dat spustit:

```
let iframeWindow = document.querySelector('#iframe').contentWindow;
if (iframeWindow.initializeTableFeatures) {
  iframeWindow.initializeTableFeatures();
}
```

Celý skript tabulky je tak zabalen do funkce, která se spouští ze skriptu uživatelské stránky.

3.1.3.6 Datová tabulka

Skript tabulky je odpovědný za vyhledávání, řazení a stahování tabulky ve formátech PDF, JSON, CSV a XLSX. Funkce searchTable reaguje na uživatelský vstup do vyhledávacího pole, iteruje přes všechny řádky tabulky a rozhoduje o jejich viditelnosti porovnáním textu v každém řádku s hledaným výrazem. Řádky, jejichž text neobsahuje hledaný výraz (nezávisle na velikosti písmen), jsou skryty přidáním třídy hide. Každému řádku je také nastavena CSS proměnná --delay pro

animační efekty. Nakonec se aktualizuje barva pozadí viditelných řádků, střídavě na průhlednou nebo lehký odstín šedé, pro zlepšení čitelnosti. Dále kód také umožňuje řazení tabulky kliknutím na záhlaví sloupců, přičemž při každém kliknutí se mění směr řazení. Při aktivaci se zvýrazní odpovídající sloupec a záhlaví, a následně funkce `sortTable` seřadí řádky vzestupně nebo sestupně na základě textového obsahu buněk v daném sloupci. Před řazením je nutné pomocí `spread` operátoru vytvořit nové pole z `NodeListu` získaného z `DOM`, který ale neumožňuje metody pole jako je `sort`.

```
function sortTable(column, sort_asc) {
  [...table_rows].sort((a, b) => {
    let first_row = a.querySelector('td')[column].textContent.toLowerCase(),
        second_row = b.querySelector('td')[column].textContent.toLowerCase();
    return sort_asc ? (first_row < second_row ? 1 : -1) : (first_row <
second_row ? -1 : 1);
  }).map(sorted_row =>
document.querySelector('tbody').appendChild(sorted_row));
}
```

Funkce na konverzi tabulky fungují v závislosti na vybraném formátu, po konverzi dat ale využívají všechny funkce společný mechanismus pro vytvoření souboru a jeho stažení. To se děje pomocí vytvoření HTML elementu `<a>`, který obsahuje data a atribut pro nastavení názvu souboru. Následně je tento element programově "kliknut" pro iniciování stahování a poté odstraněn z dokumentu. Formát elementu `<a>`:

```
const mime_types = {
  'json': 'application/json',
  'csv': 'text/csv',
  'excel': 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'}
a.href = `data:${mime_types[fileType]};charset=utf-8,${encodeURIComponent(data)}`;
```

3.1.3.7 Ošetření a výjimky na klientovi

Chybové hlášky fungují v celé webové aplikaci na stejném principu, to znamená server pošle v odpovědi klientovi status a případně chybovou zprávu, kterou následně JavaScript zobrazí ať už v podobě textu na stránce, nebo alert hlášení. V kódu to vypadá takto:

```
data.success ? fetchUsers() : alert('Chyba při mazání uživatele.');
```

3.1.3.8 ReCaptcha na klientovi

Verifikace pomocí Captcha testu funguje na všech úrovních webové aplikace. HTML se připojuje na Google API rozhraní, které má reCaptchu na starosti. Ve formuláři na přihlášení nebo registraci se pak nachází klientský klíč stránky.

```
<div class="g-recaptcha" data-sitekey="6LeLG1QpAAAAEBuKsb-  
RaFSAkTsgq2gTWId0FMZ"></div>
```

JavaScript zkontroluje, zda je Captcha vyplněná, o to, jestli je vyplněná správně se pak stará PHP.

```
const captchaResponse = grecaptcha.getResponse();  
if (!captchaResponse.length > 0) {  
    document.querySelector('#errorMsg').innerText = 'Vyplňte prosím captcha  
test.';  
    document.querySelector('#errorMsg').style.display = 'block';  
}
```

3.2 Backend

Serverová část projektu se stará o zpracování dat a správnou a bezpečnou komunikaci s klientem a databází. Celý backend je psán v jazyce PHP.

3.2.1 PHP

3.2.1.1 Obecně o PHP

PHP je serverový programovací jazyk. Výhodou je, že uživatel stránky nevidí zdrojové skripty, čímž se liší například od JavaScriptu, který na druhou

stranu dokáže dynamicky reagovat na události způsobené klientem, je proto nejvhodnější tyto dva jazyky kombinovat. PHP interpreter na serveru pracuje tak, že HTML příkazy rovnou ukládá do výsledné stránky, ale pokud narazí na PHP script, nejprve ho provede a až pak je do HTML stránky zapsán jeho výsledek. Zkratka PHP původně znamenala Personal Home Page, ale postupným vývojem se nyní označuje jako hypertextový preprocesor.

3.2.1.2 Obecně o SQL

Structured Query Language, zkratkou SQL, je prostředník komunikace s relační databází. Příkazy SQL se využívají na provádění operací jako jsou aktualizace dat nebo načítání dat z databáze. Syntaxe vychází z angličtiny. Od programovacích jazyků se liší tím, že nespecifikuje jak se má něco provést, ale jen že se má něco provést.

3.2.1.3 ReCaptcha na serveru

Při pokusu o přihlášení server pošle data z reCaptcha na Google API, která vrátí data o tom, jestli byl test vyplněn správně. Až poté následuje samotná logika přihlašování či registrace.

```
$url = 'https://www.google.com/recaptcha/api/siteverify';
$captchaResponse = file_get_contents($url, false, $context);
$captchaResponseData = json_decode($captchaResponse);
if (!$captchaResponseData->success) {
    echo json_encode(['success' => false, 'message' => 'Captcha verifikace selhala']);
    exit;
}
```

Z pohledu uživatele kontrola testu probíhá už na widgetu na stránce, toto opatření slouží k tomu, kdyby někdo poslal uměle vytvořený dotaz na server.

3.2.1.4 Připojení k databázi

Všechny PHP soubory pracují se stejným PDO, což je rozhraní pro práci s SQL databází, nadefinovaným v jednom souboru. Samotné připojení:

```
try {  
    $pdo=new PDO("mysql:host=$host;dbname=$db;charset=utf8mb4", $user, $pass);  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
} catch(PDOException $e) {  
    exit("Chyba při připojování k databázi: " . $e->getMessage());  
}  
return $pdo;
```

3.2.1.5 Ošetření a výjimky

Práce s chybami probíhá na serveru zpravidla tak, že je většina kódu obalena v try catch bloku. Server jako odpověď na dotaz odesílá JSON, ve kterém se klient dozví, zda byla operace úspěšná či nikoli a případnou chybovou zprávu.

```
try {  
    $response["success"] = false;  
    if (podminka) {  
        $response["success"] = true;  
        $response["message"] = "uspech";  
    } else {  
        $response["message"] = "neuspech";  
    }  
} catch(PDOException $e) {  
    $response["message"] = "Chyba: " . $e->getMessage();  
};  
echo json_encode($response);
```

3.2.1.6 Nahrávání na server

Celý projekt závisí na správných datech v databázi a práci s nimi, začněme tedy tím, jak se data strukturují do databáze. V části dokumentace s názvem administrace webu jsem již popisoval, jak získat relativně malý soubor užitečných dat, který lze vložit do webové aplikace a teď popíšu, jak to probíhá z pohledu serveru a co se s ním děje dál.

Nejprve se metodou post na server nahraje daný archiv, který se rozbalí a soubor v něm zkontroluje, následně se vytvoří záloha databáze, smažou se stará data v databázi, to je záležitost několika SQL dotazů a následně se vloží nová data.

```
if (file_exists($lock_file)) {  
    backupDatabase($pdo);  
    $deleteMessage = deleteAllData();  
    $insertMessage = insertJsonIntoDatabase($lock_file);  
    $assignMessage = assignNearestCity();  
    $deleteFileMessage = deleteJson($lock_file);  
}
```

Nová data se vkládají tak, že se nejprve vloží kategorie do tabulky kategorií, následně objekty, ke kterému je přiřazena adresa a adrese existující město. Pokud adresa nemá existující město, což je častý případ, musí se přiřadit k objektu podle jeho souřadnic nejbližší město. To si žádá výpočet nejkratší vzdálenosti od všech měst, to odpovídá Asymptotické složitosti $O(N^2)$. Takový výpočet lze zařídit třemi způsoby, buď výpočtem geodetické vzdálenosti pomocí Haversinovy vzdálenosti, která je sice nejpresnější, ale vzhledem k počtu adres bez města, kterých může být přes deset tisíc a počtu měst, kterých se v datové sadě nachází přes pět set, by taková výpočetní operace byla extrémně náročná, a proto je nutné nepočítat se zakřivením země, které je v České republice, ze které data jsou, minimální. Zbylými možnostmi jsou Euklidovská vzdálenost a Manhattanská vzdálenost, která je nejrychlejší, protože na rozdíl od výše zmíněné nepočítá s Pythagorovou větou, ale pouze sčítá absolutní hodnoty vzdáleností na dvou osách. To, podle mých výpočtů, s sebou nese zrychlení o 58 % oproti Euklidovské vzdálenosti, na druhou stranu to ale znamená přiřazení jiného, než nejbližšího města ve 22 % případů, což je příliš vysoké číslo, a tak jsem použil Euklidovskou vzdálenost.

```
$distance = sqrt(pow($address['latitude'] - $city['latitude'], 2) +  
pow($address['longitude'] - $city['longitude'], 2));
```

Pokud se celá tato operace nezdaří, databáze se obnoví ze zálohy. Celý tento proces nahrávání nových dat je chráněn proti spuštění vícekrát v jeden moment pomocí zamykacího souboru, který se vytvoří na začátku a na konci procesu se zase odstraní. Příklad vytvoření zálohy:

```
function backupDatabase($pdo) {
    $backupTables = ['object', 'address', 'categories'];

    foreach ($backupTables as $table) {
        $backupTable = $table . '_backup';
        $pdo->exec("DROP TABLE IF EXISTS `{$backupTable}`");
        $pdo->exec("CREATE TABLE `{$backupTable}` LIKE `{$table}`");
        $pdo->exec("INSERT INTO `{$backupTable}` SELECT * FROM `{$table}`");
    }
}
```

3.2.1.7 Práce serveru s databází

V předchozí části jsem psal o funkcích `insertJsonIntoDatabase` a `assignNearestCity`, které bych chtěl podrobněji rozebrat.

3.2.1.7.1 Funkce `insertJsonIntoDatabase`

Funkce načítá data z JSON souboru, dekoduje je a vkládá do databáze. Prochází každou kategorií a objekty v JSONu, přičemž vkládá nové kategorie pomocí `INSERT IGNORE` (aby se ignorovaly duplicity) a získává jejich ID. Pro objekty řeší získání `city_id`, pokud totiž město není nalezeno, nastaví se `city_id` na 0, toho pak využívá druhá zmíněná funkce. Následně vkládá objekty s daty do tabulky `object` s přidělením unikátního ID. Adresy objektů se vkládají do tabulky `address` s odkazem na `object_id` a následně se aktualizuje záznam v `object` o `address_id`. Vše probíhá pomocí předpřipravených SQL dotazů.

```
$stmtObject = $pdo->prepare("INSERT INTO object (category_id, name, email,
phone, website, latitude, longitude) VALUES (?, ?, ?, ?, ?, ?, ?)");
$stmtObject->execute([$categoryId, $object['name'], $object['email'],
$object['phone'], $website, $object['coordinates']['lat'],
$object['coordinates']['lon']]);
$objectId = $pdo->lastInsertId();
```


3.2.1.7.2 Funkce assignNearestCity

Funkce aktualizuje záznamy adres v databázi tak, že každé adrese bez určeného city_id přiřadí nejbližší město na základě geografických souřadnic. Nejprve získá seznam všech měst a jejich souřadnic z databáze. Poté vyhledá všechny adresy, kterým chybí přiřazení k městu (city_id je NULL nebo 0), a pro každou z těchto adres vypočítá vzdálenost, o které jsem již psal. Pro každou adresu nalezne město s nejmenší vzdáleností a aktualizuje city_id adresy na ID tohoto nejbližšího města.

```
foreach ($cities as $city) {
    $distance = sqrt(pow($address['latitude'] - $city['latitude'], 2) +
pow($address['longitude'] - $city['longitude'], 2));
    if ($distance < $smallestDistance) {
        $smallestDistance = $distance;
        $nearestCityId = $city['id'];
    }
}
$stmtUpdate = $pdo->prepare(
    "UPDATE address SET city_id = :city_id WHERE id = :address_id");
$stmtUpdate->execute(
    ['city_id' => $nearestCityId, 'address_id' => $address['id']]);
```

3.2.1.8 Správa uživatelů na serveru

Administrátorská část webu, která se stará o manipulaci s uživateli funguje tím způsobem, že nahraje z databáze všechny uživatele, kteří nemají roli admin a vloží je do tabulky, která se obnovuje jednak po krátkých časových intervalech, a také po každé provedené změně, aby byla data v ní aktuální. Mazání probíhá velmi přímočaře, pomocí id uživatelů v tabulce. Podobně funguje i editace, kde se nahrávají do databáze k aktualizaci uživatele všechna data o uživateli, nejen nové údaje. Tyto údaje prochází stejným validačním procesem, jako při registraci. Pokud

se nemění heslo, tedy pokud se nové heslo shoduje se starým hashem, tak se do databáze nenahrává.

```
if ($password_or_hash == $passwordHash) { // zmena bez hesla
    $stmt = $pdo->prepare("UPDATE users SET user_name = ?, email = ?
WHERE id = ?");
    $result = $stmt->execute([$username, $email, $userId]);
} else { // zmena s heslem
    $password_hash = password_hash($password_or_hash, PASSWORD_DEFAULT);
    $stmt = $pdo->prepare("UPDATE users SET user_name = ?, email = ?,
password_hash = ? WHERE id = ?");
    $result = $stmt->execute([$username, $email, $password_hash,
$userId]);
}
```

3.2.1.9 Přihlašování a registrace na serveru

Přihlašovací a registrační systém kontroluje údaje na klientské straně i na straně serveru. Registrace na straně klienta je nejdříve zkontrolována pomocí JavaScriptové knihovny just-validate, která při každé změně zkontroluje uživatelem zadané údaje a následně je, pokud je vyplněná recaptcha a email je unikátní, pošle na server, který údaje znovu zkontroluje, ověří správnost recaptcha testu a následně se pokusí vytvořit uživatele s danými daty. Pokud se to podaří, uživatele i automaticky přihlásí tím, že jeho data přidá do session. Následně se pošle zpráva na klienta, který uživatele přesměruje na uživatelskou stránku. Samotné přihlášení funguje na stejném principu. Odhlášení je vyřešeno smazáním session a přesměrováním na odhlašovací stránku.

```
if (password_verify($password, $user["password_hash"])) {
    session_start();
    $_SESSION["loggedInUserId"] = $user["id"];
    $_SESSION['user_roles'] = $user_roles;
    $_SESSION['user_name'] = $user['user_name'];
    $_SESSION['last_activity'] = time();
    $response["success"] = true;
    $response["message"] = "Přihlášení proběhlo úspěšně";
}
```

3.2.1.10 Uživatelská část na serveru

V části hledat se nachází formulář, kde uživatel zadá kategorii a město hledaných míst. U obou funguje našeptávač, kde se klient po dvou zadaných písmenech ptá serveru POST dotazem na možné výsledky. Dále je ve formuláři vzdálenost od souřadnic města, to znamená, že server porovná vzdálenost souřadnic všech míst, které vyhovují dané kategorii, od souřadnic města. Převedením na stupně server zjistí, jestli je vzdálenost kratší, než uživatelem zadaná a následně zkontroluje filtry. Uživatel si totiž může nastavit, zda výsledky musí obsahovat email, telefon, web, a pokud se místo musí nacházet v zadaném městě. Toto zkontroluje několik podmínek a následně server vezme výsledek z databáze a vrátí jej uživateli.

```
$delta = $radius / 111.0; // Zhruba překonvertované km na stupně
$sql = "SELECT o.name, o.email, o.phone, o.website, o.latitude, o.longitude,
a.housenumber, a.street, a.postcode, c.name as city, a.country, a.suburb
FROM object o
JOIN address a ON o.address_id = a.id
JOIN cities c ON a.city_id = c.id
JOIN categories cat ON o.category_id = cat.id
WHERE cat.name = :categoryName
AND o.latitude BETWEEN (:cityLat - :delta) AND (:cityLat + :delta)
AND o.longitude BETWEEN (:cityLon - :delta) AND (:cityLon + :delta)";

// Filtry
if ($emailRequired) {
    $sql .= " AND o.email IS NOT NULL AND o.email <> ''";
}
if ($phoneRequired) {
    $sql .= " AND o.phone IS NOT NULL AND o.phone <> ''";
}
if ($websiteRequired) {
    $sql .= " AND o.website IS NOT NULL AND o.website <> ''";
}
if ($sameCityRequired) {
    $sql .= " AND c.id = :cityId";
}
```

3.2.1.11 *Role a neaktivita na serverové části*

Pokud je uživatel neaktivní, je odhlášen. Server měří čas mezi dotazy, a pokud tento čas přesáhne 15 minut, tak je mu smazána session a je přesměrován na odhlašovací stránku. Server také kontroluje roli uživatele, pokud se uživatel snaží dostat na stránky s omezeným přístupem. To se děje tak, že se porovná role v session s požadovanou rolí.

```
if ($requiredRole && (empty($_SESSION['user_roles']) ||
!in_array(strtoupper($requiredRole), $_SESSION['user_roles']))) {
    header('Location: ../login.html?insufficient_role=true');
    return false;
}
```

Závěr

Cílem projektu bylo vytvořit systém, který bude dobře hledat kontaktní údaje, a to se podařilo. Umí, co by měl umět. Při vývoji se ale vyskytlo mnoho problémů a tím největším z nich bylo získání potřebných dat a jejich vložení do databáze. Původně jsem doufal, že bude datová sada ještě bohatší, nicméně open source nenabídl tolik, co placené služby. Výhodou tak je, že se zmíněná data dají relativně snadno vyměnit, pokud by tento systém měl být reálně používán.

Při tvorbě projektu jsem nejen utvrdil získané znalosti, ale také jsem se naučil spoustu nových dovedností, které se mi budou jistě hodit. Do budoucna lze projekt rozšířit o marketingový nástroj, který bude data z tohoto systému využívat, jak už jsem zmínil v úvodu.

Seznam přístupových údajů

URL adresa webu: <https://radostfi20.mp.spse-net.cz/>

Úroveň oprávnění	Přihlašovací email	Heslo
Administrátor	admin@getcontacts.com	
Registrovaný uživatel	user@gmail.com	

Seznam použité literatury a zdrojů obrázků

Obrázky byly převzaty z různých internetových zdrojů.

Seznam použité literatury:

- [1] *HTML* [online]. [cit. 2024-03-24]. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Markup_Language
- [2] *CSS* [online]. [cit. 2024-03-24]. Dostupné z: <http://owebu.org/cze/css/index.php>
- [3] *JS* [online]. [cit. 2024-03-24]. Dostupné z: <https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk>
- [4] *JS AJAX* [online]. [cit. 2024-03-31]. Dostupné z: <https://www.itnetwork.cz/javascript/oop/ajax-v-javascriptu-zakladni-dotazy>
- [5] *JQuery* [online]. [cit. 2024-03-24]. Dostupné z: <https://it-slovník.cz/pojem/jquery>
- [6] *PHP* [online]. [cit. 2024-03-24]. Dostupné z: <https://www.builder.cz/rubriky/php/php-cast-i-uvod-do-jazyka-155594cz>
- [7] *SQL* [online]. [cit. 2024-03-24]. Dostupné z: <https://prahacoding.cz/co-je-to-sql/>
- [8] *Výpočet vzdálenosti* [online]. [cit. 2024-03-31]. Dostupné z: <https://www.id-sign.com/poradna/vypocet-vzdalenosti-z-gps-souradnic>

[9] *Výpočet vzdálenosti* [online]. [cit. 2024-03-31]. Dostupné z:
<https://portal.matematickabiologie.cz/index.php?pg=analiza-a-hodnoceni-biologickych-dat--vicerozmerne-metody-pro-analyzu-dat--podobnosti-a-vzdalenosti-ve-vicerozmernem-prostor--metriky-pro-urceni-vzdalenosti-a-podobnosti-mezi-dvema-vektory--metriky-pro-urceni-vzdalenosti-mezi-dvema-vektory-s-kvalitativnimi-hodnotami-souradnic>