UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Pracovná plocha výskumníka

Diplomová práca

2017 **Bc. Miroslav Gregorec**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Pracovná plocha výskumníka

Diplomová práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: FMFI.KAI Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Batislava 2017 **Bc. Miroslav Gregorec**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Miroslav Gregorec
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Pracovná plocha výskumníka
Researcher's Desktop

Cieľ: Cieľom práce je vyvinúť aplikáciu, ktorá slúži ako knowledge base a zároveň odkladacia skriňa pre výskumníka. Eviduje tam svoje články a prezentácie, posudky, práce, ktoré viedol, správy, pracovné cesty, kontakty, projekty, literatúru, sleduje konferencie a odbornú literatúru, vedie si zoznam svojich citácií a všetko má dostupné kdekoľvek z internetu. Všetky záznamy je možné navzájom previazať, pridávať k nim prílohy, exerné odkazy, klúčové slová a komentáre. Systém má responzívny design. Práca nadväzuje na tri predchádzajúce bakalárske práce, ktoré boli vyvíjané vo frameworku GWT, ktorých výsledkom bolo iba rozpracovanie prototypu, avšak kvôli rozsahu nad rámec bakalárskej práce sa ani raz nepodarilo dospieť k úplnému a nasaditeľnému systému. Applikáciu na základe prototypov je nutné vyvinúť od začiatku, keďže predchádzajúce verzie stavajú na menej aktuálnej technológii. Webová aplikácia naprogramovaná v Jave alebo v Javascripte/ node.js. Z odborného a teoretického hľadiska sa predpokladá, že si študent naštuduje a použije technológie informačných systémov nad rámec štúdijného programu - urobí prehľad moderných frameworkov pre aplikačné servre a jeden z nich využije.

Literatúra: Marc Harter: Node.js in Practice, Manning Publications, 2014.
Eric Jendrock at.al: The Java EE 7 Tutorial, vol1, vol2.
Ondrej Brichta: Pracovná plocha výskumníka, bakalárska práca, FMFI UK, 2014.

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 03.10.2016

Dátum schválenia: 17.10.2016 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasuje, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Bc. Miroslav Gregorec

Chcem sa poďakovať svojmu školiteľovi Mgr. Pavlovi Petrovičovi, Phd. za cennú pomoc, rady, konzultácie, ochotu a hlavne čas, ktorý mi venoval počas písania diplomovej práce.

Abstrakt

Práca sa venuje vytvoreniu webovej aplikácie, ktorá slúži ako úložisko pre výskumníka a súčasne spĺňa základné prvky organizéra. Používateľ do nej ukladá štruktúrované dáta či súbory do rôznych kategórií podľa témy a vie v nich efektívne vyhľadávať. Práca obsahuje teoretické východiská kde sa venuje voľbe vhodného nástroja a programovacieho jazyka na vývoj danej aplikácie a analyzuje full-textové vyhľadávanie a jeho použitie vo viacerých databázach s cieľom zabezpečiť efektívne vyhľadávanie vo vyvíjanom programe. Neposlednou časťou tejto práce je popis funkčných požiadaviek aplikácie, porovnanie podobných existujúcich programov a popísanie použitých technológií na implementáciu. Výsledkom práce je webová aplikácia Pracovná plocha výskumníka, obsahujúca základnú funkcionalitu popísanú v požiadavkách. Program je responzívny a ľahko modifikovateľný. Jednotlivé kategórie a ich vlastnosti sú automaticky generované z konfiguračného súboru čo zaisťuje maximálnu mieru prispôsobenia programu na potreby používateľa, bez akýchkoľvek zmien v zdrojovom kóde. Pracovná plocha výskumníka je naprogramovaná v jazyku JavaScript – Node.js, čo zabezpečuje že aplikácia má rýchlu odozvu a používa jednotný programovací jazyk na strane servera aj klienta. Aplikácia je nasadená do reálneho pracovného prostredia a svojimi funkciami zjednodušuje prácu výskumných pracovníkov.

Kľúčové slová:
Webová apikácia, JavaScript, Express, Node.js, Angular 2, Bootstrap, MySQL

Abstract

The work is devoted to creating a web application that serves as a repository for the researcher and at the same time fulfills the basic elements of the organizer. The user saves structured data or files into different categories by subject and he can search effectively for them. The work contains theoretical background where it focuses on choosing the right tool and programming language to develop this application and analyzes full-text search, and its use in multiple databases to ensure effective search in the developed program. The other part of this work is a description of the functional requirements of the application, a comparison of similar existing programs and a description of the technologies used for implementation. The result of this work is a web application The Researcher's Desktop, containing the functionality described in the requirements. The program is responsive. The categories and their properties are automatically generated from configuration file, which ensures the maximum customization of the program to user requirements, without any changes in the source code. The Researcher's Desktop is developed in JavaScript – Node.js, which ensures that the application has a quick response and uses a single programming language on both sides – server and also client. The application is deployed in a real work environment and simplifies the work of researchers by its features.

Kľúčové slová:

Web application, JavaScript, Express, Node.js, Angular 2, Bootstrap, MySQL

Obsah

1. Úvod

1.1. Motivácia a analýza problematiky

Každodenná práca výskumníka môže byť veľmi rozsiahla a náročná, vyžaduje si synchronizáciu množstva činností, efektívnu organizáciu pracovného času, či prehľadné ukladanie rôznych poznatkov ako sú knihy, časopisy, fotografie, kontakty alebo iné súbory. Avšak nie je dôležité materiály len ukladať ale hlavne rýchlo a efektívne sa k ním znovu dopracovať.

Hlavným cieľom diplomovej práce je zjednodušiť prácu výskumníka tým, že si bude môcť ukladať všetky elektronické materiály „pod jednu strechu“, a to prakticky kdekoľvek a kedykoľvek. Pracovná plocha výskumníka je teda webová aplikácia, umožňujúca štruktúrovane ukladať elektronické informácie rôzneho druhu a zároveň spĺňa aj základné úlohy organizéra a adresára (do kalendára sa dajú pridávať udalosti, vytvárať úlohy, značiť kontakty, poznámky). Táto aplikácia je špecifická hlavne tým, že umožňuje uložiť údaje do kategórií s rozličnou štruktúrou údajov a následne medzi nimi vytvárať rôzne vzťahy, pridávať komentáre, súvisiace URL odkazy a kľúčové slová. Toto všetko má následne pomôcť zrýchliť a zlepšiť výskumníkovi vyhľadávanie potrebných údajov a orientovanie sa v množstve dát. Ďalším špecifikom aplikácie je, že zoznam kategórií, ktoré je možné v aplikácii pridávať nebude konečný a pevne daný, ale v prípade potreby sa budú dať jednoducho vytvoriť nové, plne funkčné kategórie.

1.2. Existujúce riešenia

V dnešnej dobe je vcelku náročné vytvoriť jedinečnú aplikáciu, preto i v tomto prípade existuje mnoho podobných riešení, ktoré pokrývajú časť funkcionality vyvíjanej aplikácie. Tieto riešenia by sa dali na základe podobnosti s mojou aplikáciou rozdeliť do dvoch základných skupín a to organizéry a aplikácie na správu projektov alebo súborov. Dnes už máme na internete veľké množstvo rôznych organizérov, ktoré poväčšine ponúkajú štandardné funkcie kalendára, ukladanie udalostí, pripomienky a podobne. Málo však nájdeme takých aplikácií, ktoré primárne slúžia na ukladanie dát s rôznou štruktúrou a ešte zriedkavejšie možno nájsť také, ktoré by tieto dve odvetvia prepájali. Hlavnou myšlienkou diplomovej práce je prepojenie týchto dvoch segmentov, ktoré by pokrylo všetky potreby výskumníka. Pracovná plocha výskumníka teda bude mať aj štandardné vlastnosti

organizéra ako sú kalendár , udalosti, úlohy ale bude mať aj možnosť uložiť materiály alebo kontakty z danej udalosti a jednoduchým spôsobom ich k danej udalosti priradiť, aby výskumník ani po čase nezabudol ako k daným materiálom či kontaktom prišiel.

1.3. Cieľ práce

Cieľom diplomovej práce je vytvoriť webovú aplikáciu Pracovná plocha výskumníka, analyzovať a porovnať tejto aplikácie s podobnými existujúcimi riešeniami.

Súčasťou diplomovej práce je aj podrobnejšie analyzovať Node.js server a Javascript ako zvolený programovací jazyk a následne výskum a analýza používaných full-text vyhľadávaní. Po podrobnej analýze bude nutné nadobudnuté vedomosti zužitkovať v prospech vyvíjanej aplikácie a zabezpečiť tak spoľahlivosť systému a jednoduché vyhľadávanie dát. V neposlednom rade bude dôležité systém navrhnúť tak, aby ho bolo možné neskôr jednoducho upravovať a rozširovať.

1.4. Štruktúra práce

Prvá kapitola sa zaoberá úvodom do problematiky, opisom základného problému a počiatočným návrhom riešenia.

Druhá kapitola sa zaoberá teoretickými východiskami a teda podrobnou analýzou Node.js web servera a používaním JavaScriptu. Rozoberá taktiež dôvody prečo sme sa rozhodli na implementáciu použiť JavaScript a nie Javu EE.

Tretia kapitola sa venuje výskumu Full-textových vyhľadávaní, oboznámením sa s takýmto vyhľadávaním a názorným ukážkam v najpoužívanejších databázových systémoch.

Štvrtá kapitola sa zaoberá konkrétnymi používateľskými požiadavkami na aplikáciu, z ktorých vychádza funkcionalita a systémové požiadavky.

V Piatej kapitole sa venujeme prehľadu podobných riešení a analýze odlišností medzi danou aplikáciou a už existujúcimi riešeniami.

Šiesta kapitola sa zaoberá prehľadom použitých technológií pri vývoji aplikácie, návrhom používateľského rozhrania, databázy a architektúrou aplikácie..

V siedmej kapitole sa nachádza implementácia systému. Charakterizuje sa štruktúra zdrojového kódu, vysvetľuje sa základná funkcionalita a približuje sa použitie daných frameworkov.

V predposlednej kapitole sa popisujú kroky nutné k inštalácii a následnému používaniu danej vyvinutej aplikácie.

Posledná kapitola predstavuje záver a zhrnutie výsledkov práce.

2. Teoretické východiská

Nasledovná kapitola sa zaoberá bližšími špecifikáciami Node.js technológie, výhodami a dôvodom použitia tejto technológie v našej práci.

2.1. Ako funguje Node.js

Hlavným rozlišovacím znakom Nodu je jeho neblokujúca „event-driven“ architektúra, za použitia asynchrónnych I/O volaní operujúcich v jednom vlákne. Bežne používané webové servery zvládajú súbežnosť vytváraním nových vlákien pre každú novú požiadavku, čím často dochádza k zahlteniu pamäte. Node je schopný vďaka jeho unikátnym vlastnostiam podporovať desiatky tisíc súbežných pripojení aj s obmedzenou pamäťou a jediným vláknom. Node môže dosiahnuť vysokú mieru súbežnosti bez nutnosti používania iných vlákien.

Pre lepšie vysvetlenie a porozumenie architektúry Nodu, objasníme v nasledujúcich podkapitolách jeho základné funkcie a porovnáme ich s predchádzajúcimi prístupmi.

2.1.1. Neblokujúci cyklus udalostí

Nasledujúca kapitola je spracovaná podľa [8]. Node je neblokujúci v tom zmysle, že je schopný obsluhovať viacero požiadaviek na server a pritom neplytvať prázdnymi cyklami v I/O úlohách, ako je to v prípade bežných blokovacích modelov. Bežné blokovacie modely zvyčajne blokujú ďalšiu požiadavku na server, keď sa vykonáva I/O operácia ako je napríklad čítanie z databázy. Aby bol Node neblokujúci, používa cyklus udalostí – softvérový model, ktorý používa neblokujúci I/O kombinovaný s event-driven (program riadený udalosťami) I/O – ďalšia registrovaná udalosť je vyvolaná vtedy, keď sa stane niečo v programe.

```php
<?php
//Blokujúci PHP kód
print('Ahoj');
sleep(10);
print('PHP');
print('Dovidenia');

// Tento program vypíše:
// Ahoj
// (počká 10 milisec.)
// PHP
// Dovidenia

```

Obrázok 1
Príklad blokujúceho PHP kódu

Pre lepšie pochopenie príklad blokujúceho PHP kódu (obr. 1) a Node.js kódu (obr. 2).

```
1   // Neblokujúci Node.js kód
2   console.log('Ahoj');
3   setTimeout(function (){
4       console.log('Node.js');
5   }, 10000);
6   console.log('Dovidenia');
7
8   // Tento program vypíše:
9   // Ahoj
10  // Dovidenia
11  // Node.js
12
```

Obrázok 2
Príklad neblokujúceho Node.js

V prvom príklade, PHP *sleep()* funkcia blokuje vykonávacie vlákno. Kým program „spí“, nevykonáva žiadnu úlohu, ale čaká špecifický čas a zatiaľ nie sú vykonávané žiadne ďalšie inštrukcie, kým čas neuplynie. Je zjavné že tento prístup je synchrónny. Na druhej strane Node využíva cyklus udalostí a tak aj napriek použitiu blokovacieho *setTimeout()* je táto funkcia neblokovaná. Funkcia zaregistruje udalosť pre budúcnosť a dovolí programu pokračovať ďalej – preto je asynchrónna.

2.1.2. Jednovláknový model

Zatiaľ čo iné webové servery ako Apache vytvárajú pre každú požiadavku nové vlákno, Node obsluhuje všetky žiadosti za použitia cyklu udalostí a jedného vlákna. Zvažujúc okolnosti, že Node dovoľuje neblokujúcim I/O, aby sa nachádzali v jednom jadre, pričom sa Node nepreťažuje, pretože nie sú vytvárané nové vlákna, ho robí veľmi výnimočným. Keď Node aplikácia potrebuje vykonať operáciu, pošle asynchrónnu úlohu do cyklu udalostí, zaregistruje callback funkciu a ďalej pokračuje vo vybavovaní ďalších operácií. Cyklus udalostí neustále vykonáva asynchrónne operácie, spustí danú callback funkciu a po jej ukončení vráti jej výsledok do aplikácie. Node je schopný pokryť veľké množstvo operácií (dokonca aj s jedným vláknom), riadením vláknového fondu a optimalizovaním spúšťania úloh, ako sú klientské spojenie a výpočty. Táto kapitola bola spracovaná s použitím [16].

Ako príklad môžeme vidieť obrázok 3 a obrázok 4, kde používame funkciu *getJazyky()*, ktorá vracia kolekcie jazykov v html forme.

```
var jazyky = [
    'Node.js',
    'PHP',
    'C++',
    'Java'
];

function getJazyky () {
    var html = '<b>' + jazyky.join(('</b><br><b>') + '</b>');
    //resetovanie poľa
    jazyky = [];
    return html;
}
```

Obrázok 3
Príklad jedno-vláknového modelu Node.js

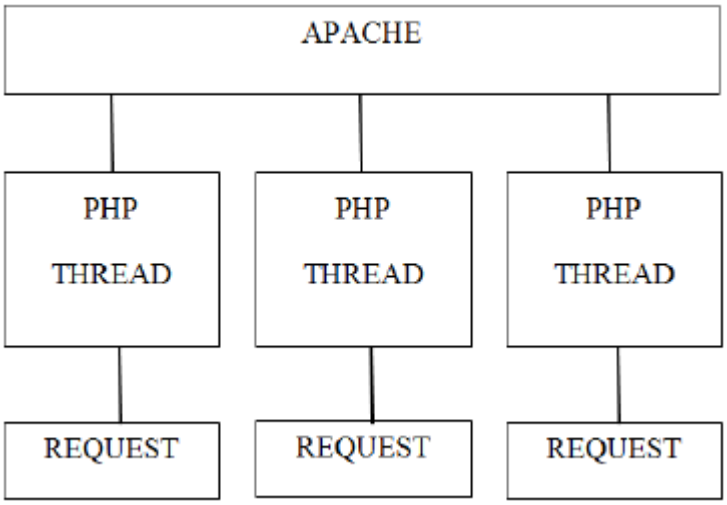
```
<?php
$jazyky = array(
    'Node.js',
    'PHP',
    'C++',
    'Java'
);

function getJazyky () {
    $html = '<b>' . join ($jazyky, '</b><br><b>') + '</b>';
    $jazyky = array();
    return $html;
}
```

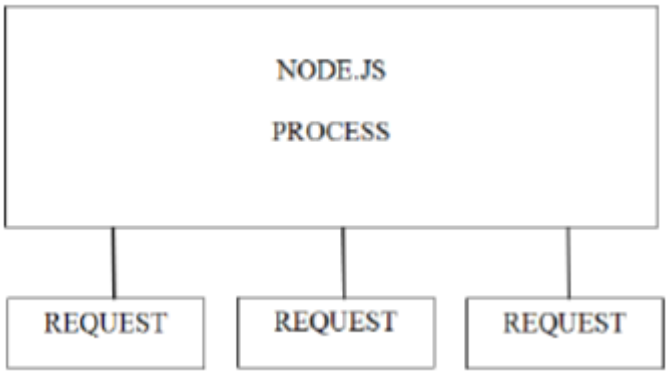
Obrázok 4
Príklad viacvláknového modelu v PHP

Na predchádzajúcich obrázkoch vidíme ekvivalentný kód v Node a PHP, avšak po každom ďalšom requste funkcie *getJazyky()* Node a PHP vracajú rozdielne výsledky. Node spracuje prvý request a vráti hodnotu zreťazených stringov jazykov v html premennej. Druhý request nevracia nič, pretože obsah premennej (html) nie je ovplyvnený, nakoľko Node pracuje v jednom procese. PHP vracia zreťazený string jazykov v oboch prípadoch, pretože premenná $*jazyky* je upravená zakaždým v novom vlákne pre každý request.

V súvislosti s predchádzajúcimi príkladmi môžeme nakresliť nasledovné diagramy: Obrázok 5 a Obrázok 6.



Obrázok 5
Vláknový model Apache-PHP [18]



Obrázok 6
Jedno-vláknový model Node.js [18]

2.1.3. Asynchrónne programovanie

Kým neblokujúca časť Nodu dovolí virtuálne prijať a spracovať všetky requesty, asynchrónne programovanie zabezpečuje, aby boli requesty zvládnuté efektívne, využitím obmedzených časových cyklov a dostupnej pamäte v svojej jednovláknovej architektúre [8]. Takmer všetky API vystavené prostredníctvom Node modulov sú asynchrónne (môžu existovať aj synchrónne verzie). Node je schopný dosiahnuť vysokej miery súbežnosti práve vďaka asynchrónnym volaniam cez callback funkciu.

Pre lepšie porozumenie konceptu asynchrónnych volaní, zvažujme tieto tri príklady čítania textového súboru na obrázkoch 7 až 9.

```php
<?php
// Verzia PHP - synchrónne čítanie textového súboru

$file = fopen('info.txt', 'r');
// počká kým sa súbor otvorí

$contents = fread($file, 100000);
// počká kým sa načíta obsah

print ($contents);
// vypíše obsah
```

Obrázok 7
Čítanie textového súboru pomocou PHP synchrónne

Na Obr. 7 je očividné, že táto verzia je neefektívna a plytvá časom používateľa robením ničoho, kým čaká ako súborový systém urobí svoju prácu.

Obrázok 8 je prepísanie synchrónnej verzie PHP kódu do Node verzie. Nakoľko je tento kód nesprávne napísaný a padá na chybe – pretože *fs.open* funkcia je asynchrónna. Premenná *file* nie je nastavená až pokiaľ súbor nie je otvorený a prijatý v callback funkcii ako tretí parameter k *fs.open* funkcii.

```javascript
// Nesprávne asynchrónne čítanie súboru s Node.js

var fs = require('fs');
var file;
var buf = new Buffer(100000);
fs.open(
    'info.txt', 'r',
    function (handle) {
        // treti argument je callback funkcia
        file = handle;
    }
);

fs.read(
    file, buffer, 0, 100000, null,
    function () {
        console.log(buf.toString());
        file.close(file, function () {});
    }
)
```

Obrázok 8
Čítanie textového súboru pomocou nesprávneho asynchrónneho Node.js

Táto verzia kódu na obr. 9 má callback funkciu podanú ako tretí parameter asynchrónnej funkcie(*fs.open*). Prvý parameter callback funkcie označuje úspešnosť aj neúspešnosť

poslednej operácie a druhý parameter označuje dodatočné výsledky, alebo informácie z poslednej operácie.

```
1   // Spravne asynchrónne čítanie súboru s Node.js a o3etrn9m chyb
2
3   var fs = require('fs');
4
5   fs.open(
6       'info.txt','r',
7       function (err,handle) {
8           if(ee){
9               console.log("ERROR: " + err.code
10                          + " (" + err.message + ") ");
11              return;
12          }
13          var buf = new Buffer (100000);
14          fs.read(
15              handle, buf, 0, 100000, null,
16              function(err,length) {
17                  if (err) {
18                      console.log("ERROR: " + err.code
19                          + " (" + err.message + ") ");
20                      return;
21                  }
22                  console.log(buf.toString('utf8', 0, length));
23                  fs.close(handle, function() {})
24              }
25          )
26      }
27  )
```

Obrázok 9
Čítanie textového súboru pomocou asynchrónneho Node.js

2.2. Vysoko-výkonnostný Web-server – Node.js

So vznikom Web 2.0 a neobmedzeným internetovým pripojením množstva rozličných zariadení ako sú telefóny, tablety či notebooky, sa rapídne zvýšil aj rozsah aplikácií. Vyrovnať sa problému s veľkým nárastom užívateľov a dodať zážitok v reálnom čase, sa stalo hlavným problémom. Kým inštalovanie nového hardweru kladie dôraz na neustále sa zvyšujúce požiadavky a rýchlosť pripojenia (čo nie je optimálne, pretože to zvyšuje cenu), Node začal tento problém riešiť predstavením novej architektúry zvanej „event-driven programming“ pre webové servery [6]. Node je mnohokrát účinnejší pri práci s pamäťou než zaužívané servery a je schopný poskytovať veľmi rýchlu časovú odozvu aj pre viacerých používateľov súčasne. Tieto výhody sú zabezpečené vďaka tomu, že Node beží na jednom vlákne a vždy keď musí urobiť niečo pomalé, ako napríklad počkať na potvrdenie, sa jednoducho posunie na inú požiadavku. Konvenčné prístupy pri takomto probléme využívajú viaceré vlákna, ktoré vyžadujú viacej pamäte.

2.2.1. Popularita JavaScriptu

Od začiatku vývoja WWW, bol JavaScript súčasťou prehliadačov. Aj napriek odlúčeniu AJAXu je JavaScript dôležitý, nakoľko je to jediný programovací jazyk, ktorý štandardne funguje na všetkých prehliadačoch. Odhliadnuc od toho, aký programovací jazyk je použitý na serveri, bol Javascript vždy vhodnou voľbou pre programovanie klientskej časti. Časté používanie Javascriptu a lipnutie Nodu na tomto jazyku, s možnosťami vývoja servera a klienta v tom istom programovacom jazyku a ďalšími inými výhodami, doviedlo vývojárov k osvojeniu si Nodu. Vďaka využitiu vlastností JavaScriptu ako jazyka a starostlivosti o užívateľskú komunitu, sa Node stal veľmi obľúbenou platformou s pokračujúcim rastom používateľov [7].

2.2.2. Jeden jazyk – viacero využití

Node umožňuje používanie JavaScriptu na strane klienta aj servera, čím posunul JavaScript na vrchol pri vývoji aplikácií. Ľubovoľný systém vyvíjaný v Node, nemá problém fungovať hocikde na lokálnom serveri, klientskej platforme, či high-end Node serveri hosťovanom kdekoľvek. Naviac je množstvo modulov(rozšírení), ktoré sú voľne dostupné. Na rozdiel od ostatných programovacích jazykov, kde je pri vývoji či testovaní potrebný web server, Node má všetko čo web server vyžaduje (client-side scripting, server-side scripting) a preto môže byť aplikácia vyvíjaná lokálne s použitím Node built-in web servera[7].

2.2.3. Jednoduché vývojárske prostredie

Nastavenie a rozbehanie vývojového prostredia pri viacerých projektoch, býva väčšinou pre vývojárov ťažkopádne a otravné. Správne nastavenie prostredia, uistenie sa, že všetky balíky sú nainštalované správne a v správnych verziách a následné uloženie kódu do repozitára na pretestovanie, zaberá množstvo času a námahy. Častokrát sa musia programátori vracať späť ku krokom, na ktoré zabudli, či ich vynechali. Node tento proces značne zjednodušuje, čo zvyšuje pracovnú produktivitu. Vývojár si jednoducho Node stiahne, nainštaluje a už ho môže používať vo svojom programe aj v lokálnom prostredí. Inštalácie Node sú dostupné pre všetky hlavné operačné systémy ako: Mac, Linux, Windows a SunOs.

2.3. Prehľad frameworkov pre aplikačné servery

V nasledujúcej kapitole charakterizujeme niektoré z najpoužívanejších frameworkov pre tvorbu webových aplikácií v Java2 EE a Node.js. Nasledujúce podkapitoly sú spracované na základe zdrojov [19] [20] [21].

2.3.1. Spring MVC

Spring MVC je aplikačný Java2 EE/Java framework. Ponúka užitočnú sadu nástrojov pre vývoj a konfiguráciu webových aplikácií. Spring je medzi programátormi známy svojim dobre rozvinutým ekosystémom. Má množstvo doplnkov, ako sú služby SOAP, API , REST, či rôzne bezpečnostné overovania .

Výhody:

- Vylepšená modularita pre lepšiu čitateľnosť kódu
- Zjednodušená injection pre testovanie údajov prostredníctvom POJO
- Flexibilné používanie Dependency Injection (DI)
- Jednoduché používanie rôznych modulov

Zápory:

- Pre neskúseného vývojára môže byť zložitý na implementáciu

2.3.2. JSF – (JavaServer Faces)

JSF je Java/Java2 EE framework s podporou od Oracle. Tento framework síce nemusí byť najlepším, ale vďaka dokumentácii dodanej spoločnosťou Oracle, sa ľahko používa. Framework obsahuje množstvo nástrojov a knižníc, ktoré vedia vypomôcť aj pri komplexných aplikáciách. JSF využíva rôzne serverové stránky Javy ale podporuje aj iné technológie ako Facelets a XUL.

Výhody:

- JSF je dôležitou súčasťou Java EE a ako taký je vhodný pre vývojárov, ktorí používajú softvér IDE

Zápory:

- JSF je zložitý na pochopenie bez predchádzajúcich zručností s vývojom Java web aplikácií

2.3.3. GWT (Google Web Toolkit)

GWT je open source balík nástrojov pre vývoj webaplikácií. GWT kompiluje celý kód určený pre klientské rozhranie do jazyka JavaScript, aby zabezpečil jeho optimalizáciu. GWT sa od predchádzajúcich dvoch frameworkou odlišuje tým, že používa architektúru MVC(Model-View-Controler), ale používa MVP (Model-View-Presenter).

Výhody:

- Jednoduchý na naučenie
- Dobré použitie najmä pri aplikáciách s väčším zaťažením na strane klienta
- Zabudovaná podpora IDE na priame refaktorovanie Java kódu

Zápory:

- Kompilácia GWT je pomalá
- Častá zmena verzií GWT

2.3.4. Express.js

Express.js je ľahký výkonný middleware a routing framework. Je jedným z najznámejších node.js frameworkov. Express je schopný vytvoriť http server v Node, middleware štruktúru a request/response funkcie potrebné na fungovanie webovej aplikácie.

Výhody:

- Veľká komunita užívateľov
- Jednoduché spravenie bežných webových aplikácií

Zápory:

- Neposkytuje infraštruktúru na väčšie projekty
- Použitie balíkov tretích strán môže spôsobiť problémy s kompatibilitou

2.3.5. Hapi.js

Hapi je ďalším frameworkom Node.js. Pôvodne bol napísaný na základoch Express, ale v neskorších verziách bol Express framework z neho úplne odstránený. Aj napriek určitým podobnostiam medzi týmito frameworkami , Hapi je primárne určený na tvorbu a testovanie APIs (application programming interfaces).

Výhody:

- Bežné užívateľské prípady a potreby sú súčasťami frameworku, čo znamená menej práce pre vývojárov
- Štruktúra Hapi aplikácie je vhodná na prácu v tímoch

Zápory:

- Malá komunita používateľov

2.3.6. Sails.js

Sail.js je ďalším MVC frameworkom, skvelým pre aplikácie s vysokou náročnosťou, postavená na platforme Express.js a Socket.IO. Je ideálny na písanie data-oriented aplikácií a real-time aplikácií (napr. chatovanie, hry).

Výhody:

- Komplexný balík Sails zahŕňa pipeline management, Websockets, ORM, security policies a rôzne iné balíčky, takže zájdenie mimo projektu Sails je pri bežných projektoch veľmi neobvyklé

Zápory:

- Slabá dokumentácia
- Slabá miera prispôsobivosti

2.3.7. Zhrnutie a výber vhodného frameworku

Na porovnanie sme zvolili niekoľko frameworkov používajúcich jazyk Java, alebo JavaScript teda Node.js. Výber frameworku pre použitie v našej aplikácii bol ťažký, keďže každý framework má svoje výhody. Po dôkladnom zvážení sme sa rozhodli použiť framework Express.js a to najmä z dôvodu, že je pomerne jednoduchý na naučenie, má veľkú užívateľskú podporu a keďže je to jeden z prvých Node.js frameworkov, má aj silné a stabilné zázemie.

3. Fulltextové vyhľadávania

Nakoľko sa má aplikácia Pracovná plocha výskumníka líšiť od ostatných aplikácií aj v tom, že uložené údaje sa budú dať jednoducho a hlavne rýchlo vyhľadať pomocou fulltextového vyhľadávania.

3.1. Čo je Fulltextové vyhľadávanie

Full Text Search (FTS) je technika na vyhľadávanie informácií v dokumentoch, textových dátach, alebo v textových databázach. Vyhľadávacie kritériá sú použité na analýzu dát a vyhľadanie všetkých zhodujúcich sa výsledkov, v prípade potreby ich zoradenie podľa relevantnosti k vyhľadávaniu.

3.2. Indexovanie

Pri práci s menším množstvom údajov, je možné pre FTS vyhľadávanie priamo skenovať obsah textu s každým dopytom. Takýto postup sa nazýva „serial scanning“.

Ak ale máme väčšie množstvo textu na vyhľadávanie, alebo je vysoký počet dopytov, je FTS často delené do dvoch úloh: indexovanie a vyhľadávanie. Fáza indexovania naskenuje text a vytvorí z neho zoznam hľadaných výrazov. Indexer vytvorí záznam pre každé slovo nájdené v texte a zaznačí si aj jeho relevantnosť k textu a následne vo fáze vyhľadávania, keď sa vyhľadáva určitý dopyt, sa hľadanie vykonáva už len v indexoch a nie v celom texte.

Predtým ako charakterizujeme konkrétne príklady fulltextového vyhľadávania, vysvetlíme nasledujúce dôležité výrazy, ktoré sa používajú pri FTS.

- **Stop Words –** sú to irelevantné slová, ktoré by mali byť vyfiltrované z textu. Napríklad: spojky, predložky, zámená
- **Stemming –** je to process redukcie slov na ich základ (stem). Napríklad slová ako lístok, list, listnatý, listina - všetky tieto slová majú rovnaký základ
- **Scoring –** rebríček merania, ktorý z hľadaných výsledkov je najrelevantnejší
- **Výraz** – slovo/slovné spojenie vyskytujúce sa v prehľadávanom texte

3.3. Full text vyhľadávanie v MySQL

Ako konkrétny príklad si rozoberieme ako funguje fulltextové vyhľadávanie v MySql. MySql využíva pri bežnom FTS vyhľadávaní ranking a vector spaces [10].

Rank, často nazývaný aj relevance rank je číslo, ktoré nám hovorí, aká dobrá je zhoda hľadaného slova a dopytu.

Vector space, nazývaný aj „natural language“, je systém založený na metaforických bodoch, ktoré sa rozkladajú do viacerých rozmerov (jeden rozmer pre jeden výraz), vzdialených od seba rôznou vzdialenosťou (jedna jednotka vzdialenosti pre jeden výskyt výrazu). Výhoda tohto spôsobu usporiadania je, keď si uvedomíme, že výskyty výrazov sú body vo viacrozmerných priestoroch, pričom v týchto priestoroch môžeme použiť základné vlastnosti trigonometrie na výpočet vzdialenosti týchto bodov, ktorá predstavuje meranie podobnosti daných výrazov.

Miera podobnosti výrazov je definovaná ako:

$$w = tf \times idf$$

Kde miera podobnosti (váha - w) je definovaná ako súčin frekvencie výskytu výrazov v dokumente (dokument - predstavuje indexované časti riadkov v databáze) a frekvencie výskytu výrazov vo všetkých dokumentoch.

Príklad: ak sa slovo „jazero“ objaví 3 krát v dokumente #5 váha mu narastie, ale ak sa slovo „jazero“ taktiež objaví v 1000 ďalších iných dokumentoch, jeho váha klesne.

MySQL používa upravený variant tohto klasického vzorca a pridáva k nemu niekoľko výpočtov pre normalizačný faktor. Po celkovej úprave vzorec vyzerá nasledovne:

$$w = \frac{log(dtf) + 1}{sumdtf} \times \frac{U}{1 + 0.0115 \times U} \times log(\frac{N - nf}{nf})$$

Kde:

dtf – počet koľkokrát sa výraz objavil v dokumente

sumdtf – je suma (log(dtf)+1) pre všetky výrazy v jednom dokumente

U – počet unikátnych výrazov v dokumente

N – počet všetkých dokumentov

Nf – počet dokumentov, ktoré obsahujú výraz

Vzorec má 3 časti: základná časť, normalizačný faktor a global multiplier.

Základná časť je ľavá strana formuly: *(log(dtf)+1)/sumdtf*

Normalizačný faktor predstavuje strednú časť formuly. Hlavná idea normalizácie je: ak je dokument kratší ako priemerná dĺžka, tak vzdialenosť ostáva rovnaká, ak je dlhší ako priemer, tak sa vzdialenosť znižuje. MySql používa unikátny normalizačný faktor. Znamená to, že merania dokumentovej dĺžky sú založené na unikátnych výrazoch v dokumente.

Ak vynásobíme základnú časť s normalizačným faktorom, dostaneme vzdialenosť výrazu, a teda tú hodnotu, ktorú MySQL ukladá do indexov.

Global multiplier je finálnou časťou vzorca. V klasickom Vector Space vzorci by to bola inverzná početnosť, teda:

$$log(\frac{N}{nf})$$

Táto časť bola nahradená:

$$log(\frac{N-nf}{nf})$$

Tento variant sa častejšie využíva v pravdepodobnostných vzorcoch. Pri týchto vzorcoch sa snaží vytvoriť lepší odhad pravdepodobnosti, že výraz bude relevantný k vyhľadávaniu.

Rank je súčinom vzdialenosti(váhy - w) a početnosti slova v query:

$$R = w \times qf$$

Kde:

w – je vzdialenosť (váha)

qf – číslo koľkokrát sa výraz vyskytne v query

Vo vektorovom priestore sa hovorí, že podobnosť je súčin vektorov. A R je číslo s pohyblivou desatinnou čiarkou, ktoré môžeme vidieť ak použijeme dotaz ako: *SELECT MATCH(...) AGAINST (...) FROM t*.

Najpodstatnejšie je teda w, čo je skratka pre vzdialenosť (váhu), ktorá sa dvíha ak sa výraz vyskytuje častejšie po sebe a spadne ak sa vyskytuje vo veľa dokumentoch. Pohybuje sa hore/dole v závislosti, či je počet unikátnych slov v dokumente menší/ väčší ako priemer. Potom R, čo je skratka pre Rank, je w-násobok frekvencie výrazu v dokumente a výrazu v query.

3.4.MongoDB Text Search

Použitím MongoDB FTS, môžeme definovať textové indexy na ľubovoľné pole v dokumente, ktorého hodnota je string alebo pole stringov. Keď vytvoríme textový index na pole, MongoDb vytvorí z poľa stem a stokenizuje ho a následne nastaví index.

Detailné spracovanie môžeme vidieť v nasledujúcom príklade, v ktorom sme vložili do databázy vetu „*dogs who cats dont eat ate rats*“

```
"parsedTextQuery" : {
        "terms" : [
                "dog",
                "cat",
                "dont",
                "eat",
                "ate",
                "rat",
                "dog",
                "eat"
        ],
        "negatedTerms" : [
                "friend"
        ],
        "phrases" : [
                "dogs eat"
        ],
        "negatedPhrases" : [ ]
 }
```

Obrázok 11
Ukážka z databázy MongoDB [11]

Následne keď máme v databáze indexy vo viac ako v jednom poli, väčšinou sa stáva, že nie všetky polia sú rovnako dôležité ako ostatné. Predvolená hodnota pre každý index poľa

je 1. Váhu indexu je možné meniť manuálne, ale samotný algoritmus ju neupravuje. Z toho vyplýva, že MongoDB vracia výsledky hľadania za normálnych okolností v neusporiadanom zozname[11].

3.5. PostgreSQL Full Text vyhľadávanie

Podobne ako to bolo aj pri MySQL aj PostgreSQL, využíva pri FTS Vector Space model (VSM) získavania informácií. V tomto modely ukladá rovnako ako MySQL informácie o vlastnostiach výrazov, ako napríklad: frekvencia výskytu, blízkosť, unikátnosť, atď. Výsledky hľadania a poradie podľa relevantnosti, sú stanovené porovnaním dokumentu a dotazových vektorov. V kapitole 5.3 sme sa zamerali najmä na postup váhovania jednotlivých výrazov v MySQL. V tejto kapitole sa zameriame na tokenizáciu a vytvorenie lexém z výrazov v PostgreSQL[12].

Dokument môže byť uložený v databáze vo viacerých formách. Môže využiť relačné vzťahy PostgreSQL a to napríklad byť uložený ako jeden riadok v tabuľke s viacero poľami, v jednom poli v rade, alebo v rôznych riadkoch v rôznych tabuľkách. Keď je dokument prevedený do formy vektora , typ *tsvector* ukladá celý dokument pre vyhľadávanie. Samotná konverzia dokumentu na vektor zahŕňa viacero krokov, ako parsovanie textu na tokeny, konvertovanie tokenov do lexémy(pomocou slovníkov), a optimalizované indexovanie pre rýchle vykonanie dotazu.

3.5.1. Tokenizácia

Tokenizácia je proces rozdelenia textu na tokeny. V PostgreSQL sa využíva pri štandardnom rozdelení hneď viacero rozdeľovačov (tokenizers). Jeden rozdeľovač rozdelí text podľa medzier a interpunkcie tak, že každé slovo alebo číslo je zobrané ako token. Iné rozdeľovače vedia rozlíšiť v texte URL adresy, alebo emailové adresy, z ktorých vytvoria token, zatiaľ čo ďalšie rozpoznajú xml tagy. Samozrejme je mnoho ďalších špecifických rozdeľovačov.

Keďže všetky rozdeľovače fungujú súčasne behom parsovania, môže byť jeden text rozdelený hneď na viacero tokenov. Napríklad text „all-inclusive“ môže byť štandardným parsovaním rozdelený na 3 tokeny – „all“, „inclusive“ a „all-inclusive“ – pretože existuje tiež rozdelovač, ktorý rozpoznáva slová spojené spojovacou čiarkou ako jeden token.

Po takomto spracovaní je každý token spracovaný na lexémy za použitia slovníkov.

3.5.2. Lexémy

Lexémy sú vytvorené z tokenov za použitia rôznych slovníkov pre ošetrenie rôznych variant slov ako napríklad: stopwords, stemming, synonymá a iné.

V PostgreSQL dostávame stopwords a stemming automaticky v základnom vyhľadávaní. Ale nakoľko táto databáza neumožňuje pristupovať k súborom na serveri, nie je možné používať na vytvorenie lexém vlastné slovníky, či iné šablóny. Štandardná konfigurácia na vyhľadávanie je však natoľko robustná, že by mala postačovať na všetky bežné textové vyhľadávania.

3.5.3. Vytváranie Dokument vektora

Pri vytváraní VSM v PostgreSQL sa používa funkcia *to_tsvector*. Táto funkcia zoberie daný textový dokument a konvertuje ho na vektor tak, že je uložený v databáze pod dátovým typom *tsvector*. Príklad: danú vetu *„A Awe-Inspiring Story of a Monkey And a Pastry Chef who must Succumb a Womanizer in A MySQL Convention“* uloží v nasledujúcom formáte:

```
'awe':3 'awe-inspir':2 'chef':12 'convent':21 'inspir':4
'monkey':8 'must':14 'mysql':20 'pastri':11 'stori':5
'succumb':15 'woman':17
```

Vektor pozostáva len z platných lexém (nulové sú odstránené) a pozície slov v originálnom texte sú uložené do funkcie. V prípade, že chceme, aby slová zahŕňali takisto aj váhu, môžeme im manuálne nastaviť váhu pomocou funkcie *setweight*. Váha v PostgreSQL sa značí za pomoci písmen latinskej abecedy. Dokument vektor, ktorý má nastavenú aj váhu, potom vyzerá nasledovne:

```
'awe':3A 'awe-inspir':2B 'chef':12B 'convent':21B
'inspir':4B 'monkey':8A 'must':14A 'mysql':20A
'pastri':11B 'stori':5B 'succumb':15A 'woman':17B
```

3.6. Preklepy a nesprávne zapísané slová

Väčšinou očakávame pri dotazoch na štruktúrované dáta, ako napríklad dátumy alebo ceny, že vyhľadávanie vráti presné zhodujúce sa výsledky. Avšak dobré fulltextové

vyhľadávanie nemá tie isté obmedzenia. Namiesto toho môžeme vyhľadávanie rozšíriť o slová, ktoré sa môžu zhodovať s hľadaným výrazom a zoradiť tieto slová podľa relevantnosti k hľadanému výrazu. Teda slová, ktoré sa zhodujú najviac, zobraziť na vrchných priečkach.

V skutočnosti fulltextové vyhľadávanie, ktoré zobrazuje len údaje zhodujúce sa úplne, by bežných používateľov vôbec neuspokojilo. Na takéto vyhľadávanie sme zvyknutý aj z používania google, kedy nám vyhľadávač sám navrhne opravu nepresností v slovách, alebo nám ponúkne podobné výsledky vyhľadávaniu.

Ak sa dokument presne zhoduje s hľadaným dotazom, mal by sa zobraziť na vrchole vo výsledkoch hľadania a slabšie zhody môžu byť zahrnuté nižšie vo výsledkoch. Ak sa úplne nezhoduje žiadny dokument, prinajmenšom môžeme používateľovi ukázať aspoň potenciálne správne výsledky, ktoré v konečnom dôsledku môžu byť práve tie, ktoré používateľ hľadal.

Vo vyššie spomínaných kapitolách sme hovorili o normalizovaných tokenoch, stemmingu, synonymách, ale všetky tieto prístupy predpokladajú, že hľadané slovo je napísané správne, alebo že je iba jeden spôsob ako toto slovo povedať.

Fuzzy párovanie umožňuje vyhľadávanie nesprávne zapísaných slov priamo v čase vyhľadávania, zatiaľ čo fonetické tokenové filtre môžu byť použité pre zvukové prispôsobenie slov[13].

3.6.1. Rozmazanosť (Fuzziness)

Fuzzy párovanie označí dve slová, ktoré sú si „rozmazane“ podobné, ako keby boli tie isté slová. Najskôr ale zadefinujeme čo vlastne „rozmazanosť“ znamená[18].

V roku 1965, Vladimir Levenshtei vytvoril Levenshteinovu vzdialenosť, ktorá meria počet jednoznakových úprav potrebných na premenu jedného slova na druhé. Navrhol tri typy jednoznakových úprav:

- Zámena jedného písmena za druhé (_f_ox → _b_ox)
- Vloženie nového písmena (sic → sic_k_)
- Vymazanie písmena (b_l_ack → back)

Neskôr k nim Frederick Damerau pridal ešte jeden typ:

- Transpozícia dvoch susedných znakov (_st_ar → _ts_ar)

Napríklad pre konvertovanie slov „bieber“ na „beaver“ je nutné použiť tieto kroky:

1. Zámenu v za b : bie_b_er → bie_v_er
2. Zámenu a za i : b_i_ever → b_a_ever
3. Transpozíciu a a e : b_ae_ver → b_ea_ver

Tieto tri kroky teda reprezentujú Damerau-Levenshtein editačnú vzdialenosť 3.

Samozrejme slová „bieber“ a „beaver“ sú od seba príliš vzdialené na to aby sa dali považovať sa omyl v písaní. Damerau zistil pozorovaním, že 80% ľudských omylov pri písaní má editačnú vzdialenosť 1. Inými slovami, 80% omylov môže byť opravených jednou editáciou na pôvodné slovo.

Pre príklad Elasticsearch databáza podporuje maximálnu editačnú vzdialenosť 2. Samozrejme, že vplyv na túto vzdialenosť má aj dĺžka daného slova. 2 editácie na slove „hat“ môžu vytvoriť slovo „mad“, ktoré má úplne iný význam a teda povolenie 2 editácií na 3 písmenovom slovo je značne prehnané.

3.6.2. Fonetické zhody

V predchádzajúcej podkapitole sme sa snažili párovať niečo, hocičo, čo sa dalo vyhľadávať pre slová, ktoré zneli podobne, aj keď ich pravopis sa odlišoval.

Pre vytváranie fonetickej reprezentácie zo slov existuje niekoľko algoritmov[13]. Soundex algoritmus sa však považuje za jeden zo základných algoritmov na fonetickú reprezentáciu. Mnoho ďalších algoritmov, ako Metaphone alebo Double Metaphone, na ňom zakladajú. Soundex je fonetický algorimus na indexovanie názvov podľa zvuku, tak ako sú vyslovované v angličtine. Hlavným cieľom algoritmu je, aby sa homofónne hlásky kódovali do rovnakej reprezentácie, aby mohli byť spojené aj napriek malým odlišnostiam v pravopise. Soundex je najrozšírenejším fonetickým algoritmom (sčasti preto, že je štandardnou súčasťou databáz ako DB2, PostgreSQL, MySQL, SQLite, MS SQL Server, Oracle, a iných).

4. Systémové požiadavky

V tejto kapitole rozoberieme systémové požiadavky na aplikáciu, jednotlivé kategórie a funkcionality, ktoré bude nutné implementovať.

4.1. Užívatelia systému

Cieľovou skupinou užívateľov aplikácie sú predovšetkým ľudia, ktorí potrebujú zhromažďovať a organizovať väčšie množstvo informácií rôzneho pôvodu, efektívne v nich vyhľadávať, značiť pôvod týchto informácií, vlastné komentáre a popritom mať prehľad o dôležitých akciách a plánovaných udalostiach. Primárne je aplikácia vytváraná pre základné potreby vedeckých pracovníkov (výskumníkov), avšak vo všeobecnosti sa vďaka svojej prispôsobivosti dá využiť aj v širšom okruhu používateľov.

4.2.Funkčné požiadavky a kategórie

Aplikáciu bude tvoriť viacero kategórií, obsahujúcich záznamy s vopred definovanou štruktúrou. Štruktúra bude jedinečná pre každú kategóriu, s výnimkou tých typov údajov, ktoré budú spoločné pre každý záznam. K týmto údajom patria poznámky, kľúčové slová a zoznam URL adries a zoznam interných odkazov. V nasledujúcej kapitole vykonáme bližší rozbor spomínaných kategórií.

4.2.1. Kľúčové slová

Ku všetkým kategóriám sa dajú pridávať kľúčové slová – takzvané štítky. Zoznam štítkov si používateľ vytvára sám, na základe svojej potreby. Pri pridávaní nového záznamu si používateľ vyberie z už vytvoreného zoznamu kľúčových slov len takú podmnožinu, ktorá vystihuje daný obsah, alebo vytvorí nový štítok.

Kľúčové slová sú tvorené stromovou štruktúrou, ktorá je vhodná na zadelenie do skupín a podskupín, čím sa umožní lepšie a prehľadnejšie zadefinovanie jednotlivých štítkov k záznamom a následne aj jednoduchšie vyhľadávanie konkrétnych záznamov.

4.2.2. Poznámky

Takisto ako štítky, tak aj poznámky je možné pridávať ku všetkým kategóriám. Pri každom zázname je textové pole, do ktorého je možné vložiť novú poznámku, alebo editovať existujúcu.

4.2.3. Interné odkazy

Možnosť pridávať interné odkazy do kategórií je jednou z najdôležitejších vlastností v našom programe. Vďaka tejto vlastnosti má používateľ možnosť prelinkovať na pridávaný záznam v danej kategórii ľubovoľný, už existujúci záznam, pričom má možnosť pridať k tomuto prelinkovania svoj komentár. Napríklad k záznamu o projekte môžeme prelinkovať autora z kontaktov, posudok z kategórie posudky, udalosti z kalendára, alebo rôzne ďalšie záznamy z iných kategórií.

4.2.4. Zoznamy URL adries

Do každej kategórie má používateľ možnosť pridať URL Adresy rôznych stránok a pripojiť k nim vlastný komentár. Na odkazy sa dá po uložení záznamu prekliknúť cez daný komentár, ktorý by mal výstižne vystihovať dôvod vloženia daného odkazu (napríklad môžeme do podujatia vložiť stránky a odkazy na organizátorov).

4.2.5. Ukladanie súborov

Keďže naša aplikácia má slúžiť aj ako úložisko pre výskumníka, do každej kategórie môže používateľ nahrať súbor. Množstvo nahrávaných súborov je obmedzené len možnosťami servera. Typ súborov v aplikácii nie je definovaný, takže používateľ môže ukladať fotografie, dokumenty, alebo rôzne iné súbory.

4.2.6. Zoznam a popis základných kategórií

Základné kategórie, ktoré bude mať používateľ vytvorené hneď pri prvom použití aplikácie budú nasledovné:

- Publikácie
- Projekty
- Posudky
- Knihy a časopisy
- Konferencie
- Podujatia a pracovné cesty
- Evidencia majetku
- Softvér
- Kontakty
- Kalendár
- Úlohy
- Zápisník
- Video a fotografie
- Ostatné

Publikácie

Táto kategória bude slúžiť na ukladanie rôznych publikácií. Samotnú publikáciu bude možné uložiť v rôznych formátoch, ako napríklad pdf, .docx, zip, či iné. K publikáciám je možné ďalej ukladať štandardné údaje, ako sú názov, autor, rok vydania, vydavateľstvo, číslo vydania, či kľúčové slová. Okrem tých možností má výskumník možnosť označiť publikáciu ako vlastnú, alebo prelinkovať sem ktorékoľvek ostatné súvisiace kategórie, ako napríklad kontakty, podujatie, softvér, alebo ostatné.

Projekty

Tak isto ako v kategórii publikácie, aj do kategórie projekty sa dajú nahrávať súbory rôzneho typu. Okrem toho sa sem dajú uložiť informácie, ako je názov a popis projektu. Ďalšie pridané funkcie, ako dátum začatia a ukončenia projektu, sa dajú prepojiť s funkciou kalendára, tak ako aj autori projektu, sa môžu prepojiť s existujúcimi kontaktami. K projektom je možné linkovať aj posudky, ktoré charakterizujeme v nasledujúcom odseku. Ďalej sa dajú k projektu uložiť aj informácie o finančných nákladoch a zdrojoch financovania. K finančným nákladom sa môžu rozpisovať a konkrétne položky výdavkov.

Posudky

Kategória posudky je úzko spätá najmä s projektami. Väčšinou ide práve o posudky k projektom, ktoré autor nevytváral sám, ale samozrejme sa posudky dajú prepojiť aj k ostatným kategóriám. Obsahom tejto kategórie je štandardne názov, textové pole na napísanie posudku. V prípade že používateľ nechce písať posudok priamo v aplikácii, má možnosť aj nahratia súboru. Po prelinkovaní tejto sekcie ku kalendáru, vie používateľ vytvoriť pripomienku, kedy má byť posudok písaný.

Knihy a časopisy

V tejto časti aplikácie si používateľ môže ukladať elektronické knihy, časopisy, alebo iné rubriky. Používateľ môže uložiť nasledovné záznamy: názov, autora alebo vydavateľstvo, číslo vydania, dátum vydania a typ literatúry (kniha, časopis, rubrika, iné).

Podujatia a pracovné cesty

V tejto kategórii si výskumník ukladá všetky plánované podujatia a pracovné cesty. Ide o jednu z menej rozsiahlych, no o to podstatnejších kategórií. Ukladajú sa sem nadchádzajúce, ale aj už ukončené akcie. Používateľ si uloží dátum akcie, čo mu pomáha efektívnejšie naplánovať pracovný čas a vždy pred začatím podujatia dostane upozornenie

na začiatok podujatia. Pred, ale i počas podujatia je možné ukladať sem všetky získané a potrebné materiály, prepojiť sem novonadobudnuté kontakty, alebo pripomienkovať dôležité poznámky, či naplánovať a prezerať harmonogram podujatia.

Evidencia majetku

Používateľ si v tejto sekcii vie ukladať všetok majetok, ktorý vlastní, alebo chce vlastniť. Táto sekcia slúži najmä ako prehľad materiálu, ale takisto vďaka svojim vlastnostiam pomáha používateľovi napríklad pri reklamácii tovaru, či nájdení požičaného tovaru, pretože v evidencii majetku je možné okrem štandardných vecí, ako sú názov alebo cena, uložiť napríklad aj kópiu nákupného dokladu , faktúru či meno človeka, ktorému je daný majetok zapožičaný.

Softvér

Táto sekcia slúži na prehľad používaného softvéru. Používateľ sem môže ukladať názvy programov, účel používania, produktové kľúče, odkazy na stiahnutie či iné vlastné pripomienky.

Kontakty

Pri každodennej činnosti výskumníka, či iného vedeckého pracovníka, ktorý sa denne stretáva s množstvom ľudí je samozrejmé, že sekcia kontakty v tejto aplikácii nemôže chýbať. Ku kontaktom bude môcť používateľ ukladať širokú škálu informácií a to napríklad štandardné informácie, ako meno, priezvisko, telefóny, email, adresu, ale napríklad aj kópiu vizitky, fotografiu osoby, alebo pracovné zaradenie. Nevyplnené položky sa v programe nebudú zobrazovať.

Kalendár

Asi nikomu netreba podrobnejšie vysvetliť funkciu kalendára. Ani ten sa vo vyvíjanej aplikácii nebude nijako výrazne odlišovať od klasického kalendára a plánovača. Hlavnou súčasťou však bude prepojenie kalendára so službou Google kalendár, s automatickou synchronizáciou úloh.

Zápisník

Do sekcie zápisníka si môže používateľ značiť akékoľvek svoje poznámky, svoje ciele do budúcnosti, alebo robiť si prehľady svojich splnených úloh, či jednoducho používať

sekciu zápisník ako svoj diár alebo denník. Sekcia zápisníka je poňatá veľmi všeobecne, preto si sem môže používateľ pridávať záznamy rôzneho typu s vlastnými názvami.

Video a fotografie

V poslednej predvolenej skupine si používateľ môže ukladať všetku potrebnú foto a video dokumentáciu. Fotografie je možné členiť do albumov pre ľahšie triedenie a vyhľadávanie. Následne je možné albumy, fotografie a videá pomenovať, priradiť im lokalitu ľudí spomedzi kontaktov, či jednoducho označiť ich kľúčovými slovami.

4.3. Základné charakteristiky systému

Prístup, jednoduchosť, prispôsobivosť. Táto aplikácia je určená najmä pre ľudí, ktorí sú pracovne vyťažení a preto pri sebe potrebujú jednoduchého a efektívneho pomocníka a to kedykoľvek a kdekoľvek. Z toho vyplýva, že je potrebné vytvoriť webovú aplikáciu, ktorá môže byť prístupná kdekoľvek s pripojením internetu, bez nutnosti akejkoľvek inštalácie, a zároveň je potrebné zabezpečiť minimálnu komunikáciu klienta so serverom, aby aplikácia pracovala čo najplynulejšie, aj pri pomalom pripojení na internet.

Ďalšou požiadavkou je zabezpečiť, aby bola aplikácia jednoduchá na ovládanie, teda mala aj priateľský dizajn a intuitívne sa v nej dalo pracovať bez podrobnejšieho študovania manuálov.

Taktiež jednou z dôležitých požiadaviek je, aby sa aplikácia dala prispôsobiť čo najlepšie potrebám používateľa. Musí byť teda navrhnutá tak, aby bolo možné čo najjednoduchšie pridávať nové, či odoberať staré kategórie. Nie je nutné, aby bolo umožnené takéto zmeny robiť priamo v užívateľskom prostredí, no mali by sa dať spraviť jednoducho a bez veľkých zásahov do hlavného kódu.

4.4. Previazanosť záznamov

Táto vlastnosť patrí medzi jednu zo základných vlastností v systéme. Jedná sa o možnosť prepojenia ľubovoľných dvoch záznamov medzi sebou či už pri vytváraní nového záznamu, alebo pri editácii. Táto vlastnosť napomáha používateľovi pri lepšom uchovaní vzťahov a súvislostí medzi záznamami, ale aj pri vyhľadávaní jednotlivých záznamov.

4.5. Vyhľadávanie

Vyhľadávať v aplikácii je možné viacerými spôsobmi.

Prvým je celoslovné vyhľadávanie v celej aplikácii, teda aplikácia vyhľadáva zhodu textu v ktorejkoľvek časti aplikácie.

Druhá možnosť je vyhľadávanie vo zvolenej kategórii aplikácie. V tomto prípade program vyhľadáva iba zhodné záznamy v danej kategórii.

Tretia možnosť je vyhľadávanie cez kľúčové slová a poznámky. Program po vyhľadávaní kľúčových slov zobrazí všetky záznamy s rovnakými kľúčovými slovami, alebo poznámkami.

4.6. Zálohovanie a obnova systému

Zálohovanie nebude prebiehať na pravidelnej báze, no systém bude obsahovať funkciu zálohovania. Používateľ si teda bude môcť kedykoľvek stiahnuť a uložiť celú systémovú databázu, ktorá by sa v prípade zničenia alebo spadnutia systému dala použiť na jeho obnovu, pričom by všetky funkcie ostali zachované.

4.7. Logovanie

Aplikácia automaticky vytvára systémový log, ktorý okrem chybných hlásení obsahuje aj informácie o činnosti, teda akcie ktoré boli v systéme vykonané spolu s dátumom a časom. Log nebude za normálnych okolností sprístupnený bežnému užívateľovi, bude však slúžiť najmä správcovi systému na vylepšenie aplikácie.

5. Podobné existujúce riešenia

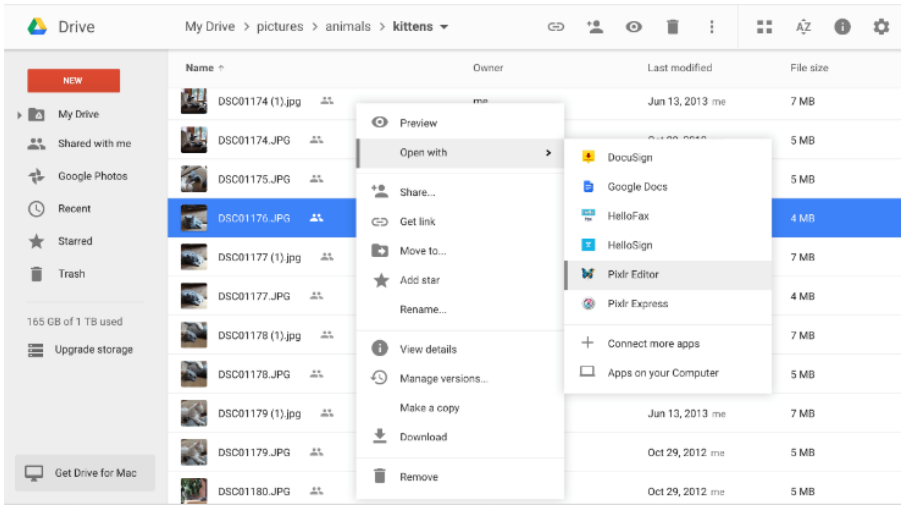
5.1. Bakalárske práce

Diplomová práca Pracovná plocha výskumníka sa opiera o predchádzajúce dve bakalárske práce z rokov 2011 a 2014, ktoré boli vyvíjané vo frameworku GWT a ktorých výsledkom bolo iba rozpracovanie prototypu aplikácie, avšak kvôli rozsahu nad rámec bakalárskej práce, sa ani raz nepodarilo dospieť k úplnému funkčnému systému. Aplikáciu je však nutné na základe prototypov vyvinúť od začiatku, keďže predchádzajúce verzie stavajú na menej aktuálnej technológii [5].

5.2. Google Drive

Google Drive je jedna s najpoužívanejších aplikácií s podobnou funkčnosťou. Slúži na ukladanie a synchronizáciu dát. Dovoľuje používateľovi ukladať dáta na cloud, synchronizovať ich naprieč zariadeniami a zdieľať ich s inými používateľmi. Pod Google Drive sa taktiež zaradujú aj Google Docs, Sheets and Slides. Sú to kancelárske nástroje, ktoré umožňujú prácu s dokumentami, tabuľkami či prezentáciami. Okrem toho sa dajú na Google drive ukladať aj fotografie či videá. Ďalšou vecou, ktorou sa zhodouje s naším systémom je prepojenie s google kalendárom, ktorý taktiež patrí do súčasti Drivu [1].

Najväčšou nevýhodou tohto systému je, že používateľ má voľných len 5GB priestoru a keď chce ukladať viac ako toto množstvo, musí platiť mesačné poplatky. Ďalšou nevýhodou je fakt, že človek odovzdáva všetky svoje informácie a súkromné fotografie spoločnosti, ktorá ich ďalej spracúva minimálne na vytvorenie cielenej reklamy.



Obrázok 12
Google Drive – užívateľské prostredie[1]

Hlavnou odlišnosťou vyvíjanej aplikácie bude systém prepájania jednotlivých záznamov, teda vytváranie vzťahov medzi záznamami, čo v Google Drive úplne chýba.

5.3.Dropbox

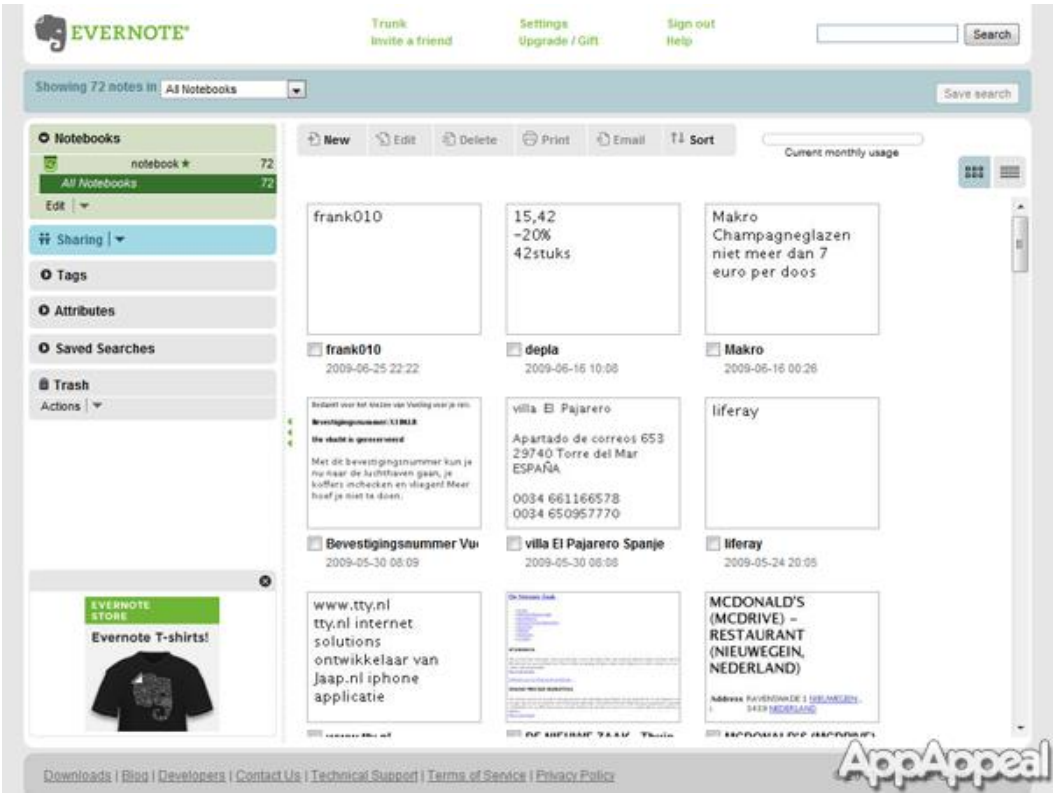
Dropbox je jedno z ďalších populárnych webových úložísk. Jeho klientské prostredie umožňuje uložiť akýkoľvek súbor a tým ho synchronizovať aj na ostatné zariadenia.

Obrázok 13
Dropbox – užívateľské prostredie [2]

Aj keď Dropbox funguje ako úložisko súborov, je zamerané hlavne na synchronizáciu a zdieľanie súborov. Drobbox je možné používať z inštalovaného klienta, alebo aj priamo z webu. Samozrejme, webový klient má určité obmedzenia (napríklad pri nahrávaní veľkých súborov). Dropbox môže slúžiť aj na zálohovanie dát, napríklad pri strate mobilného telefónu nemusíte pri automatickom zálohovaní prísť o žiadne svoje súkromné dáta [2]. Najväčšou nevýhodou je, z nášho pohľadu, len 2 GB voľného priestoru v bezplatnej verzii. Pre rozšírenia tohto množstva sú potrebné mesačné poplatky.

5.4. Evernote

Používatelia si môžu v tejto aplikácii uložiť takmer všetko, vrátane webstránok, fotiek, obrazoviek či rôznych textových poznámok. Aplikácia sa zameriava aj na to, aby k nej používateľ mohol pristupovať prakticky kdekoľvek, či už z webového prehliadača, alebo aj z mobilnej aplikácie. Takisto ako pracovná plocha výskumníka, aj v Evernote sa záznamy dajú organizovať na základe kľúčových slov, či indexov. Evernote dáva používateľovi pomerne dobré možnosti prispôsobenia aplikácie. Obsahuje niekoľko možností, ako uložiť poznámky či iné súbory. Jednou zo zaujímavostí tejto aplikácie je, že používateľ si môže zvoliť v aplikácii svoje usporiadanie záznamov, alebo môže nechať aplikáciu, aby to spravila za neho. Ako aj pri predchádzajúcich riešeniach, taktiež aj táto možnosť má zdarma len obmedzenú verziu a za plnú verziu, s väčším priestorom, je nutné zaplatiť [3].



Obrázok 14
Evernote – užívateľské prostredie [3]

5.5. Windows OneDrive

Windows Skydrive je online súborové úložisko. Dovoľuje používateľovi ukladať obsah, spolupracovať a zdieľať súbory. Skydrive je prístupný kdekoľvek, a to buď cez mobilnú aplikáciu, alebo cez webové rozhranie. Súbory môžu byť ukladané do troch zložiek a to

súkromné, verejné a zdieľané zložky. Skydrive, na rozdiel od predchádzajúcich aplikácií, nie je spoplatnená a ponúka 25GB voľného priestoru. Ďalšou výhodou je, že aplikáciu vyvíja Windows, a teda je možné v nej priamo vytvárať Microsoft dokumenty, ako Word, Excel, či Powerpoint [4].

5.6. Zhrnutie

V dnešnej dobe existuje mnoho aplikácií, ktoré majú niektoré časti spoločné s Pracovnou plochou výskumníka. Pri ukážkovom výbere podobných aplikácií sme sa snažili vyberať čo najznámejšie aplikácie s podobnou témou. Nasledujúca tabuľka obsahuje zhrnutie jednotlivých vybraných aplikácií, ich výhody a nevýhody.

Názov programu	Výhody	Nevýhody
Google Drive	• ukladá rôzne typy súborov • voľných 5GB priestoru • responzívny dizajn • inštalovateľná aplikácia aj webové rozhranie	• spoplatnenie • nedajú sa navzájom prepájať uložené súbory • chýbajú kľúčové slová
Dropbox	• prispôsobiteľný dizajn • zdieľanie súborov • ukladá rôzne typy súborov	• obmedzené možnosti vo voľnej verzii • chýbajú kľúčové slová
Evernote	• ukladá aj textové poznámky • automatické alebo vlastné organizovanie záznamov • mobilná aj webová aplikácia	• chýba kalendár • obmedzená voľná verzia
Windows OneDrive	• Obsahuje balíček Office • Prepojenie s HotMailom a Bingom • Dovoľuje sťahovanie .zip súborov	• Prepojenie záznamov • Organizácia záznamov • Prispôsobenie používateľovi

6. Návrh riešenia

V tejto kapitole popíšeme použité technológie, zdôvodníme ich výber a popíšeme návrh riešenia.

6.1. Použité technológie

V tejto kapitole sa budeme venovať najmä použitým frameworkom a dôvodom ich zvolenia.

6.1.1. Express

Express je Node.js framework, určený pre tvorbu web aplikácií a APIs. Je to voľný framework, dostupný pod open-source licenciou od MIT. Hlavným programovacím jazykom, ktorý sa využíva v Expresse je Javascript. Express ponúka robustnú sadu vlastností na vývoj webových a mobilných aplikácií [14]. Niektoré z jeho základných vlastností sú:

- Umožnuje nastaviť middleware na odpovedanie pre http požiadavky
- Definuje routing tabuľku, ktorá sa používa na vykonávanie rôznych akcií, založených na metóde http a URL adrese
- Umožňuje dynamicky zobrazovať HTML stránky na základe odovzdávacích argumentov do šablón (túto vlastnosť v našej práci nevyužívame nakoľko express používame len na back-end aplikácie)

6.1.2. Angular 2

Angular 2 je ďalšou verziou najpoužívanejšieho JavaScript frameworku (Angularu js) pre tvorbu komplexných webových aplikácií. Angular 2 je programovaný v TypesScripte. Programovací jazyk TypeScript vychádza z rovnakého syntaxu a sémantiky ako JavaScript. TypeScript sa kompiluje na čistý, jednoduchý JavaScript kód, ktorý beží na ľubovoľnom prehliadači. Angular 2 prišiel takmer so všetkým, čo je potrebné na tvorbu komplikovaných front-end riešení pre webové alebo mobilné aplikácie, od množstva šablón s rýchlym renderovaním cez data management, http servisy, form handling a mnohé ďalšie [15]. Hlavnými výhodami Angular 2 sú:

- Podporuje posledné ale aj staršie verzie prehliadačov a Android 4.1+
- Je to cross platform framework
- Používa mobile first prístup

- Je plne UI responzívny
- Používa server side rendering pre rýchle zobrazenia na mobiloch
- Dobre pracuje s jazykmi kompilovanými do JavaScriptu
- Používa component based prísutp

6.1.3. Bootstrap

Bootstrap je najpoužívanejši HTML, CSS a JS framework pre vývoj rezponzívnych mobile first projektov na webe. Obsahuje základné HTML a CSS šablóny pre písmo, formuláre, tlačidlá, navigáciu a iné ďalšie komponenty rozhrania. Hlavné výhody Boostrapu sú:

- Je jednoduchý na používanie
- Dajú sa s ním jednoducho vytvoriť responzívne projekty
- Dá sa ľahko prispôsobiť projektu
- Má veľkú vývojársku podporu
- Jednoducho sa integruje do projektu
- Veľké množstvo šablón

6.1.4. MySQL

MySQL je open-source relačná databáza, vydaná pod GPL licenciou, dostupná takmer na všetkých platformách. Pre prácu s databázou sa používa SQL (Structured Query Language). Hlavnými výhodami sú:

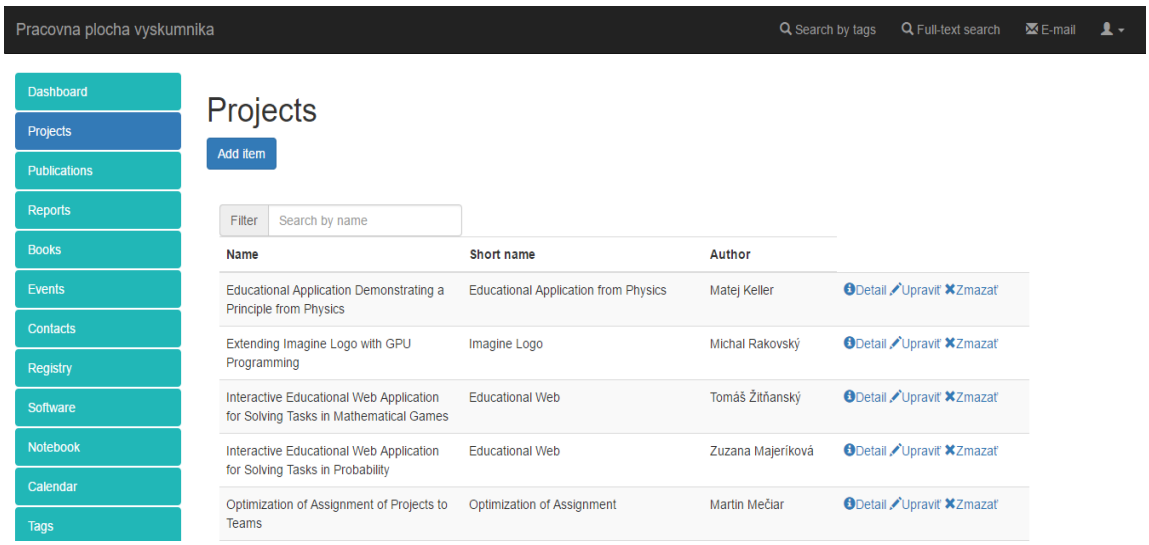
- Veľmi rýchle a spoľahlivé používanie
- Veľká vývojárska podpora
- Množstvo knižníc pre pripojenie
- Vhodná pre malé aj veľké aplikácie
- Jednoduchá inštalácia na web server

6.2.Návrh používateľského rozhrania

Dizajn aplikácie je rozdelený na 3 časti a to:

- Horná navigácia – slúži k zobrazeniu názvu aplikácie, k zobrazeniu vyhľadávačov (fulltextové vyhľadávanie a vyhľadávanie podľa tagov) a k možnosti odhlásiť sa z aplikácie a zobrazeniu detailov o užívateľovi

- Ľavá navigácia – tvorí hlavné menu aplikácie, obsahuje všetky hlavné kategórie zadefinované v programe, ďalej obsahuje prístup k nastaveniam a aj k ostatným kategóriám, ktoré nie sú generované automaticky, ako napríklad kategória domov, informácie, fotoalbum.
- Stredový panel – nachádza sa v strede obrazovky, zobrazujú sa na ňom detaily jednotlivých kategórií, teda všetky položky a záznamy. Panel sa podľa potreby konkrétnych kategórií delí následne na niekoľko menších častí.

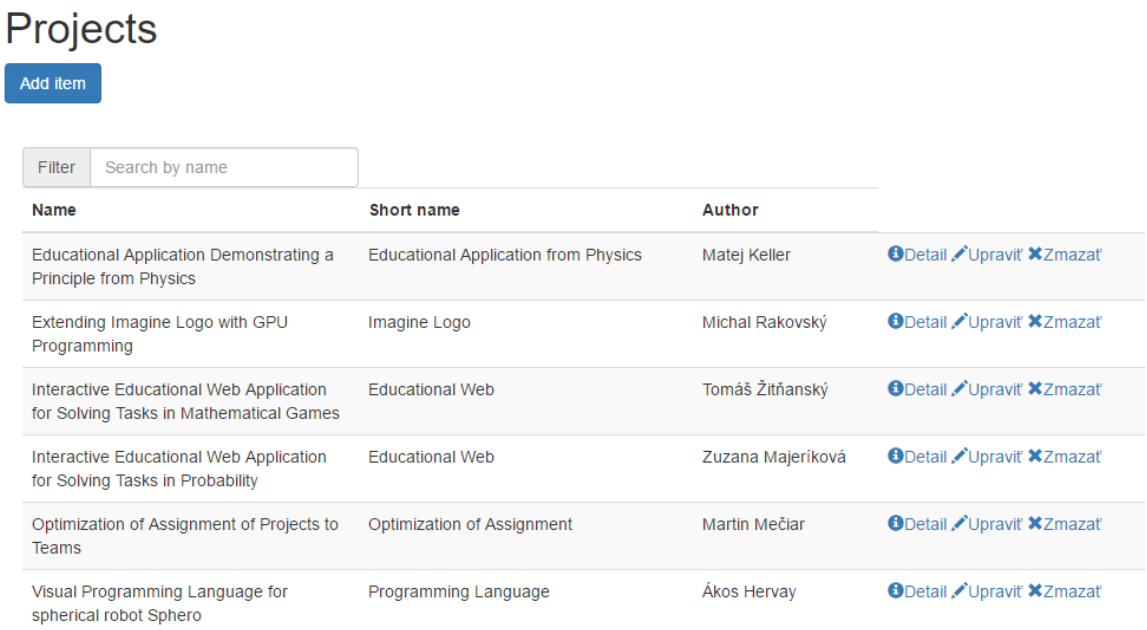


Obrázok 15
Náhľad používateľského rozhrania

6.2.1. Zobrazenie generovaných kategórií

Generované, alebo štandardné kategórie položiek sú tie kategórie, ktoré sa od seba líšia len vlastnosťami a nemajú žiadnu pridanú špeciálnu funkcionalitu. Tieto kategórie tvoria základné jadro aplikácie. Sú to napríklad: Projekty, Publikácie, Posudky, Knihy a časopisy, Podujatia a Pracovné cesty, či iné. Štandardné kategórie sa vždy zobrazujú v stredovom paneli. Pri otvorení kategórie sa nám zobrazí zoznam jednotlivých vložených položiek pod kategóriou. V zozname vidíme len základné informácie o položke, ako sú meno/ názov, skrátené meno, organizátor, autor, atď. Položky ktoré sú viditeľné sa nastavujú podľa požiadavky používateľa a bývajú rozdielne podľa kategórie. Okrem základných informácií sú na konci každého záznamu riadka tabuľky tri tlačidlá a to : *detail záznamu, úprava a zmazanie*. Funkcionalitu týchto tlačidiel je zrejme jasná z názvu, avšak zobrazenie detailu a úpravy budú bližšie špecifikované v nasledujúcich kategóriách. Mimo zoznamu položiek sa v detaile kategórie, hneď pod nadpisom, nachádza ešte tlačidlo *pridať položku*, ktoré

slúži na pridávanie jednotlivých záznamov do kategórie. Pod týmto tlačidlom sa nachádza textové pole *filter*, cez ktoré je možné fulltextovo vyhľadávať jednotlivé položky v kategórii podľa názvu.

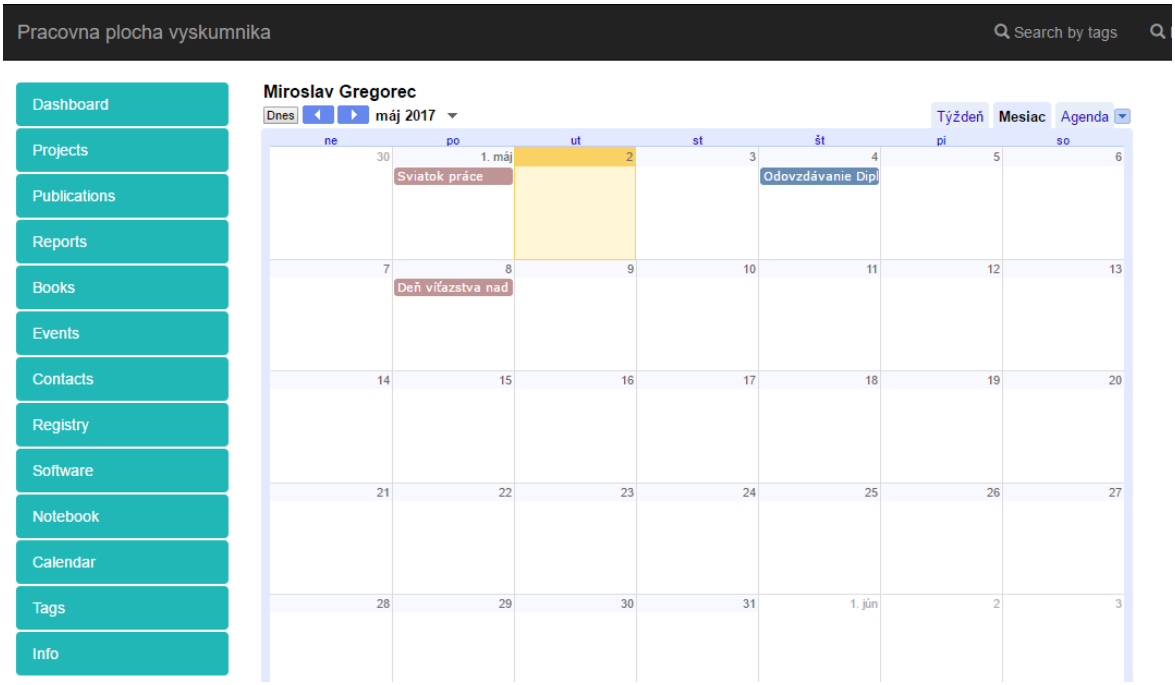


Obrázok 16
Náhľad generovanej kategórie Projects s pridanými položkami

6.2.2. Zobrazovanie špecifických kategórií

Najšpecifickejšou kategóriou našej aplikácie, je určite kategória kalendár, ale sú to aj kategórie ako napríklad: fotogaléria, info, nastavenia, alebo domov. Tieto kategórie nemôžu byť generované automaticky, pretože obsahujú špeciálnu funkcionalitu, ktorá sa líši od

bežných kategórií. Najodlišnejšia je určite kategória kalendár, ktorá predstavovala aj najzložitejšiu implementáciu, keďže nejde len o bežný vlastný kalendár aplikácie, ale o kalendár prepojený z Google api. Samotné zobrazenia kalendára, je podobné zobrazeniu kalendára cez Gmail.



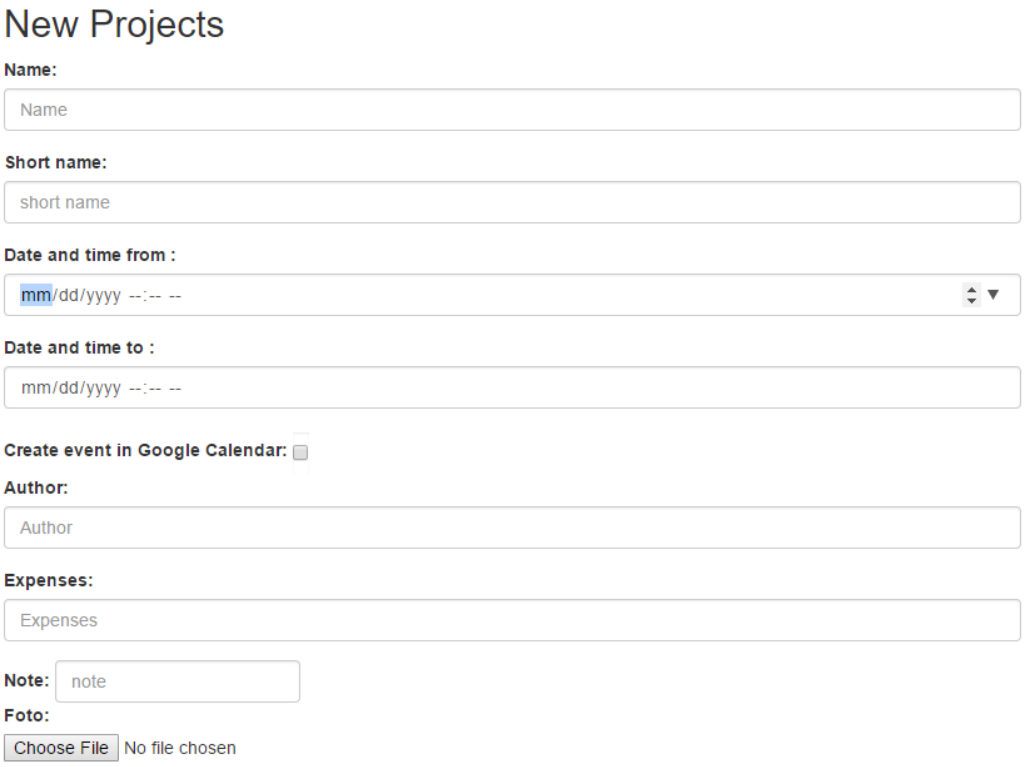
Obrázok 17
Náhľad kategórie Calendar

6.2.3. Tvorba nových záznamov a editácia

Pri tvorbe nových záznamov a editácii sa používa totožné prostredie, iba s tým rozdielom, že pri editácii sú už dáta vopred vyplnené. Údaje, ktoré používateľ vyplňuje sú konfigurovateľné a pre každú kategóriu rozdielne. V zásade sa však rozdeľujú na tieto typy:

- **Štandardné/ statické údaje -** tvoria ich údaje ako meno, názov, popis, poznámka – sú to základné údaje o zázname, ktoré používateľ manuálne vypíše do textového poľa

- **URL odkazy –** dajú sa sem ukladať odkazy z rôznych iných webových stránok, ku ktorým si používateľ vie pridať komentár. Následne po uložení sa tieto odkazy nezobrazujú celé, ale zobrazený je len komentár, s možnosťou prekliknutia sa na danú webovú adresu.

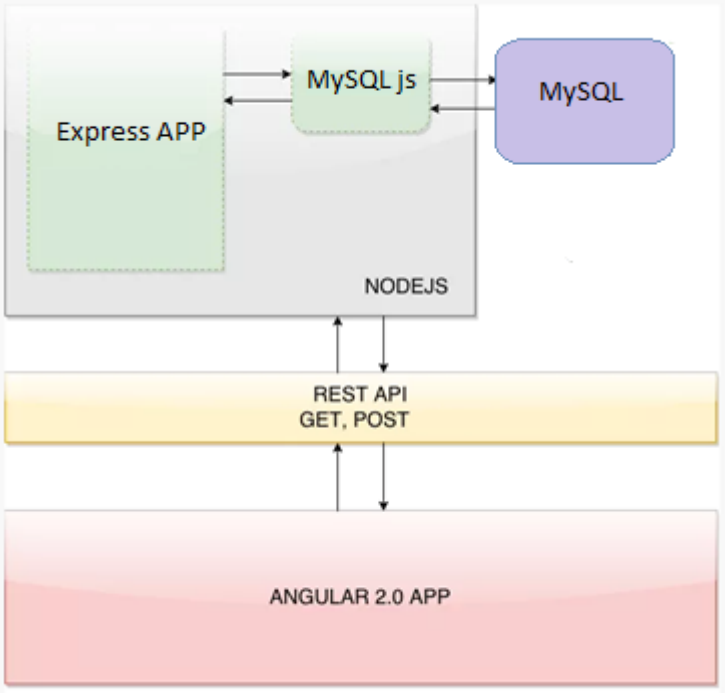


Obrázok 18
Náhľad vytvárania nového projektu

- **Interné odkazy –** v tejto sekcii má užívateľ možnosť prelinkovať k svojmu záznamu ktorýkoľvek už existujúci záznam z ľubovoľnej kategórie a pridať svoj komentár k tomuto prelinkovaniu. Napríklad môžeme prelinkovať k vytváranému projektu autora (ktorého máme uloženého v kontaktoch), posudok, či rôzne iné záznamy.
- **Tagy –** v tejto časti môže autor doplňovať k záznamu tagy(štítky) slúžiace na vyhľadávanie. V prípade že štítok ešte neexistuje, je možné ho vytvoriť a pridať k záznamu.
- **Súbory –** posledným typom dát, ktoré vieme pridávať do záznamov, sú súbory. Pod každý záznam vieme pridávať súbory rôzneho typu, či už sú to fotografie, videá, dokumenty, alebo zabalené súbory.

6.3. Architektúra aplikácie

Naša aplikácia Pracovná plocha výskumníka pozostáva, zo serverovej časti tvorenej z Nodejs a Express (express je web framework). Mysql pre ukladanie dát a Angular 2, ako klientský framework. Ako môžeme vidieť z nižšie uvedeného diagramu, Express sprístupní REST koncové body, ktoré sú používané Angularom 2., keď je spravená akákoľvek požiadavka na koncový bod REST, Express sa spojí s databázou MySQL a spracuje dáta.



Obrázok 19
Architektúra PPV

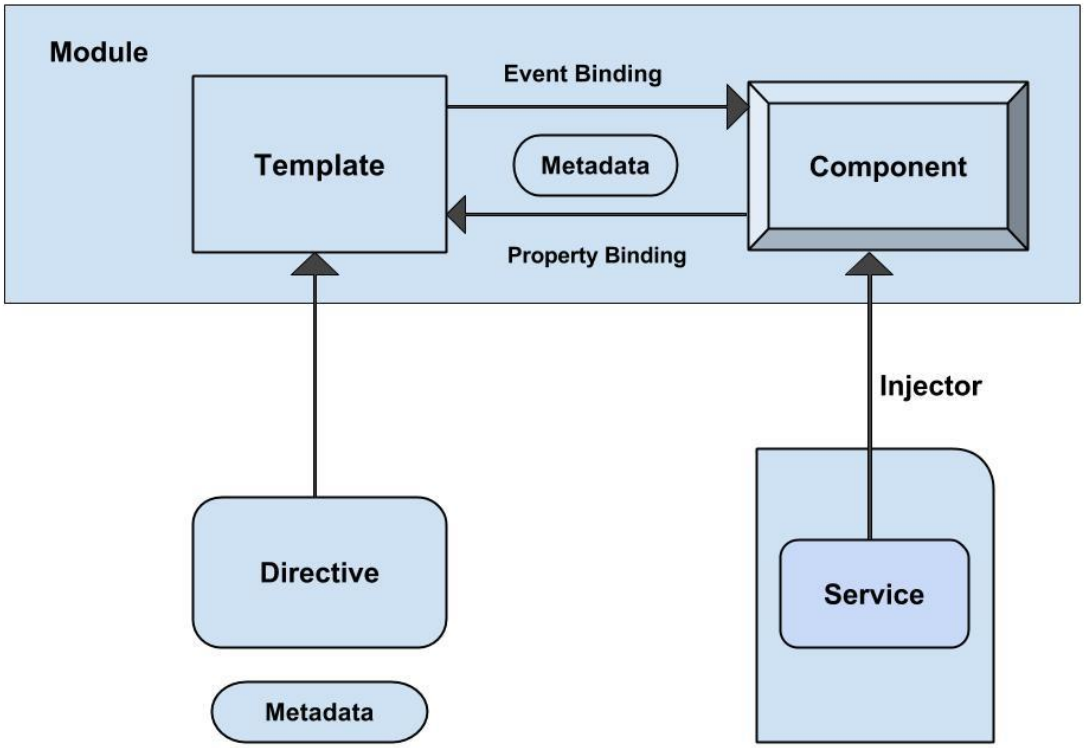
Klientskú časť aplikácie teda tvorí Angular 2, ktorého architektúra sa delí na viac modulov a to nasledovne:

- **Module (modul)** - Komponent module je charakteristický blokom kódu, ktorý môže byť použitý na vykonanie jednej úlohy. Angularové aplikácie sa nazývajú modulové, pretože k vystavaniu aplikácie používame mnoho modulov. Základným stavebným blokom Angular 2 a teda aj našej aplikácie, je *component* trieda, ktorá je exportovaná z modulu.
- **Component (komponent)** - Komponent tvorí controller trieda so šablónou, ktorá sa zaoberá najmä zobrazením aplikácie a logikou danej stránky. Je to kód,

ktorý môže byť použitý naprieč celej aplikácii. Komponent vie vykresliť sám seba a konfigurovať Dependency Injection. V Komponente môžeme pridávať CSS štýly či šablóny. V našej aplikácii máme viacero komponentov.

- **Template (šablóna)** - Zobrazenie komponentov môže byť definované pomocou šablóny, ktorá hovorí Angularu, ako zobrazovať komponenty.
- **Metadata (metadáta)** - Metadáta je spôsob spracovania triedy. Napríklad máme komponent s názvom *MyComponent,* ktorý bude triedou, pokiaľ nepovieme Angularu, že je komponent. Metadáta môžu byť pripojené do Typescriptu použitím dekorátora.
- **Data Biding (spájanie dát) –** Data Biding je spôsob koordinácie dátových hodnôt aplikácie, deklarovaním väzby medzi zdrojmi a cieľovými prvkami v HTML.
- **Service (služba) –** sú to JavaScript funkcie zodpovedné za vykonanie špecifickej úlohy. Vo všeobecnosti, je to trieda vykonávajúca špecifickú úlohu ako napríklad: prihlasovacia služba, dátová služba, atď. V Aplikácii PPV používame servisy hlavne na prepojenie frontendu a backendu – teda na získavanie dát z expressu.
- **Directive (direktíva) –** je to trieda, ktorá reprezentuje metadáta.
- **Dependency Injection (vkladanie závislostí) –** je to návrhový vzor, ktorý pracuje s objektom, ako so závislosťami v rôznych zložkách aplikácie. Vytvára novú inštanciu triedy, spolu s jeho požadovanými závislosťami.

Celková architektúra Angular 2 a teda klientskej časti našej aplikácie je nasledovná:



Obrázok 20
Architektúra Angular 2 [22]

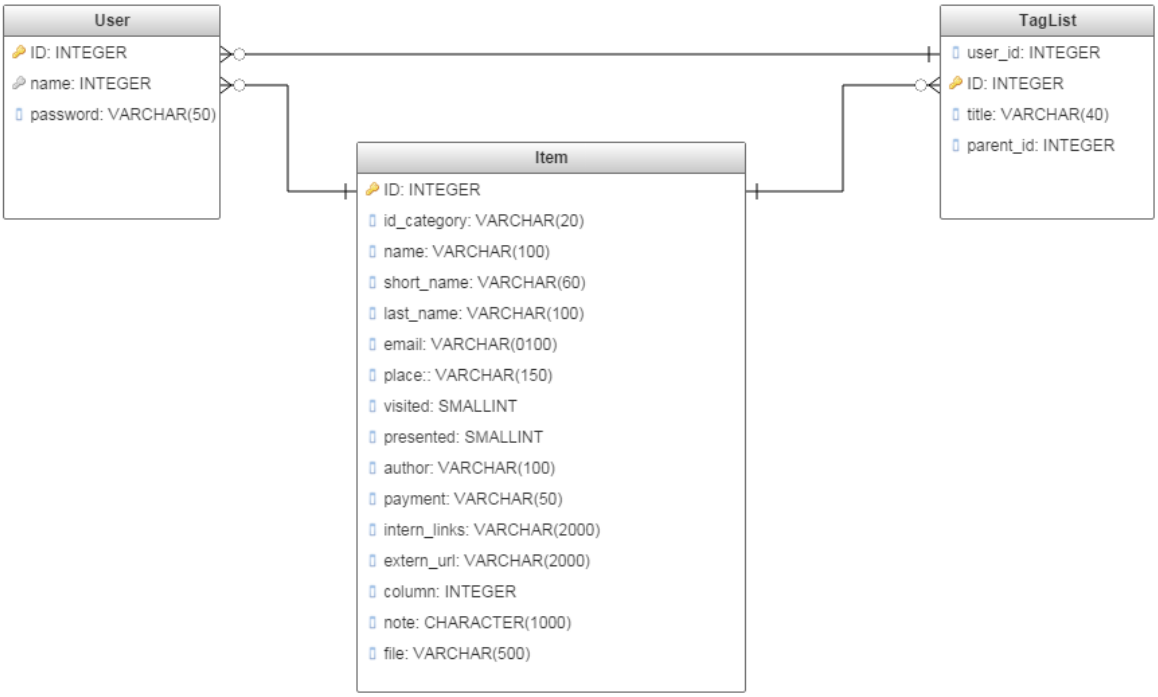
6.4. Návrh databázy

Na obrázku nižšie sú znázornené hlavné tabuľky, ktoré sa používajú v databáze našej aplikácie. Pre zachovanie prehľadnosti sme vybrali len najdôležitejšie tabuľky, ktoré sa používajú pri veľkej väčšine záznamov. Keďže väčšina kategórií je generovaná automaticky rozhodli sme sa pre použitie jednej spoločnej tabuľky, čím zachováme databázu prehľadnú a jednoduchú a to nám teda umožní jej jednoduché používanie.

Spomínanou tabuľkou je tabuľka *Item.* Ako sme už spomínali, do tejto tabuľky sa ukladajú takmer všetky záznamy z generovaných kategórií, ako sú napríklad: Projekty, Publikácie, Posudky, Knihy, Časopisy, Podujatia a iné. Tabuľka obsahuje položky ako ID - ktorá je unikátnym kľúčom každého záznamu, *id_kategory* – je kľúč danej kategórie, ďalej sa tu nachádzajú bežné fixné položky ako *name, shor_name, email, place, atď.* ktoré sú štandardne pri vytváraní záznamu, keď nie je povedané inak nastavené na null, nakoľko nie

každá kategória využíva všetky položky. Za zmienku ešte stojí položka *intern_link*, slúžiaca na ukladanie interných linkov v aplikácii, ktoré sa ukladajú ako pole objektov konvertované na *string*. Rozhodli sme sa použiť takúto formu, aby sme zamedzili zbytočným výberom z databázy a teda okrem ID linku sa ukladá aj meno, alebo názov prelinkovaného záznamu a komentár. Pri vypisovaní detailu jedného záznamu následne vykonáme len jeden Select z databázy, aj pri vypísaní viacerých interných linkov, čo nám umožní rýchlejšiu interakciu aj pri pomalšom internetovom pripojení. Takou istou položkou, ale na ukladanie externých odkazov, uchovávajúcich sa v tej istej forme, je položka *extern_url*.

Ďalšou tabuľkou v našej databáze je *TagList*. Ako sme spomínali už v systémových požiadavkách, aplikácia bude obsahovať stromovú štruktúru Tagov (kľúčových slov), ktoré budú slúžiť na jednoduchšie vyhľadávanie naprieč aplikáciou. Na ukladanie tejto stromovej štruktúry slúži práve tabuľka TagList. Obsahuje položky ID – unikátny kľúč tagu, *user_id* - je to id používateľa, ktorý daný tag vytvoril, ďalej sa tu ukladá názov tagu do položky *title* a nakoniec *parent_id* odkazujúce na rodiča tagu.



Obrázok 21
Návrh databázy PPV

Poslednou, menej dôležitou tabuľkou na diagram, je tabuľka User. Menej dôležitou z toho hľadiska, že aplikácia je vytvorená najmä na používanie pre jedného používateľa,

ktorý ju používa na vlastnom serveri a teda nie je špecifikované používanie viacerých užívateľov súčasne. Pre prípad cudzieho vniknutia, má však aplikácia plnohodnotnú autentifikáciu a teda používateľ sa musí prihlasovať cez meno a heslo, ktoré sú uchovávané v tejto tabuľke.

7. Implementácia

7.1. Sumarizácia použitých nástrojov a knižníc

Ako sme už spomínali v prechádzajúcich kapitolách, na tvorbu aplikácie PPV bolo použitých viacero frameworkov. Základom na strane servera je Express. Najpoužívanejšou knižnicou, doinštalovanou do expresu je Mysql, ktorá nám umožňuje prístup a prácu s databázou. Na strane klienta používame Angular 2, ktorý nám pomohol vytvoriť základnú funkcionalitu aplikácie. Pre vykonávanie zložitejších úkonov, ako napríklad nahrávanie súborov, filtrovanie, zobrazovanie chybných hlásení a podobne, sme používali knižnice pre Angular 2 a to napríklad: ng2-file-upload, ng2-search-filter, angular2-flash-message či iné. Okrem týchto knižníc sa pri samotnom zobrazení stránok používal bootstrap ako CSS šablóna pre jednoduchý a responzívny dizajn. Ďalej sme pri použití Google kalendáru využili Google web api, ktoré je bližšie popísané v kapitole 6.1. Použité Technológie.

7.2. Nastavenie servera a klienta

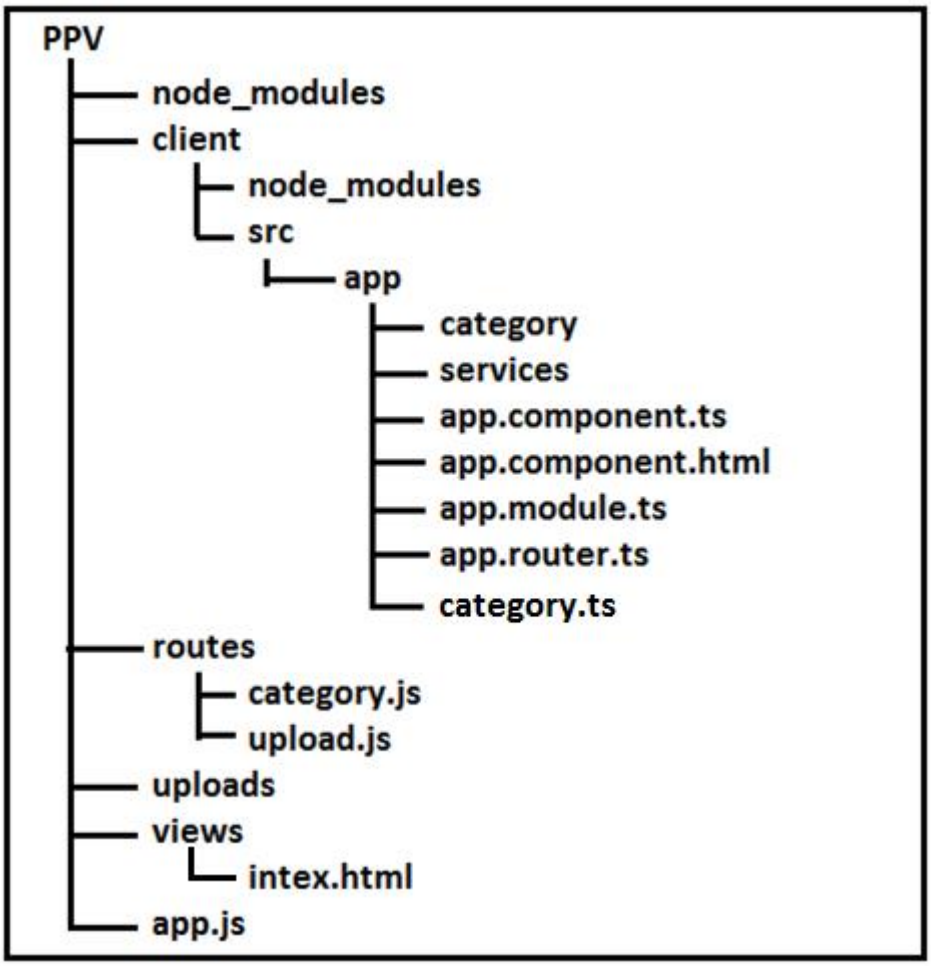
Ako prvý krok pri tvorbe aplikácie PPV, sme museli najskôr nainštalovať Node.js a npm. Inštalácia týchto komponentov nie je zložitá a na windowse prebieha štandardným spôsobom, spustením inštalačného súboru.

Ďalším naším krokom bola inštalácia frameworku Express pomocou Express generátora, ktorý nám pomohol spraviť počiatočné nastavenia pre aplikáciu. Express generátor zabezpečí všetky potrebné kroky k tomu, aby Express aplikácia bežala s Node.js. Keďže na strane klienta používame Angular 2 namiesto inštalovania šablón, vymažeme zbytočné súbory z adresár „views“, aby sme sa zbavili prebytočného kódu. Ďalej vytvoríme „client“ adresár pre všetky súčasti klientskej časti Angular 2 a pridáme tento adresár do kódu v app.js tak, aby tu Espress videl všetky statické adresáre.

Vnútri adresára client sme nastavili Angular 2. K tomu, aby sa nám to podarilo, sme použili Angular CLI. Angular CLI je nástroj na inicializáciu, vývoj a údržbu Angularových aplikácií. Po nainštalovaní Angularu je nutné presunúť súbor „index.html“ do adresára „views“ v serverovej časti aplikácie. Pre samotné počiatočné spustenie je nutné ešte spraviť build Angular 2, cez commandlinový príkaz pomocou Angular CLI a následne potom je už možné aplikáciu otvoriť vo webovom prehliadači.

7.3. Štruktúra zdrojového kódu

Ako sme už viackrát spomínali, aplikácia sa skladá z dvoch základných častí a to client (front-end) časť a server (back-end) časť. Teda je to zdrojový kód, ktorý sa vykonáva na strane klienta vo webovom prehliadači a na serveri. Tomuto rozdeleniu je prispôsobená aj celková štruktúra zdrojového kódu.



Obrázok 22
Štruktúra základných súborov aplikácie PPV

Na obrázku môžeme vidieť niektoré dôležitejšie súbory a adresáre našej aplikácie, ktoré si v tejto kapitole charakterizujeme.

node_modules - tento súbor sa nachádza v časti klienta aj servera. V týchto súboroch sa ukladajú všetky použité knižnice a závislosti Expressu a Angularu. Tieto súbory sa vytvárajú automaticky pri npm install.

app.js – je to základný konfiguračný súbor Expressovej aplikácie. Nastavujú sa v ňom všetky podstatné informácie, ako renderovací engine, závislosti, požiadavky, port pod

ktorým aplikácia funguje, či cesty aplikácie (v našom prípade pre back-end – teda prístupové body na http požiadavky).

routes – v tomto adresári sa ukladajú jednotlivé súbory podľa spracovania ciest. Sú to v podstate servisy k jednotlivým prístupovým bodom na backende. Najpoužívanejším súborom v tomto adresári je **category.js,** ktorý sa stará o prácu s databázou pre všetky generované kategórie. Spracúvajú sa tu požiadavky na výber záznamov, ukladanie, mazanie, alebo úprava. Ďalším dôležitým súborom je **upload.js,** zabezpečujúci ukladanie a mazanie pridávaných súborov na server.

uploads – v adresári uploads sa ukladajú všetky súbory, ktoré používateľ aplikácie nahráva do programu

views – ako sme už spomínali v kapitole 7.2., nastavenie Servera a klienta, v adresári view sme zmazali všetky súbory a presunuli sme do neho len súbor z Angularu 2 a to **index.html**. V tomto súbore sa deklaruje html dokument spolu s hlavičkou, titulkom a ostatnými štandardnými údajmi.

Dostávame sa ku charakteristike klientskej časti, tvorenú adresárom client. Adresár client obsahuje viacero súborov a adresárov, ale najdôležitejší z nich je adresár src, v ktorom sa nachádza jadro našej aplikácie. Ako sme si už písali v kapitole 6.3. Architektúra aplikácie, práve v tomto adresári sa nachádzajú angularové moduly a to komponenty a šablóny.

app.component.ts – robí základnú inicializáciu klientskej časti aplikácie, spolu so súborom **app.component.html** zabezpečuje renderovanie ľavého menu

```
43      {
44        path:'category/:category/:id',
45        component: DetailCategoryComponent,
46      },
```

Obrázok 23
Ukážka cesty pre detail záznamu z generovaných kategórií

app.router.ts - v tomto súbore máme definované celý klientský routing našej aplikácie, avšak väčšinu kategórií generujeme automaticky a z toho vyplýva, že aj väčšina ciest je definovaných ako premenných. Môžeme to vidieť na kóde vyššie, kde je jedným príkazom cez premenné definovaná cesta pre záznamy rôznych kategórií.

7.4. Definovanie kategórií

Zoznam kategórií nie je pevný a ako bolo povedané vo funkčných požiadavkách, má sa dať ľahko zmeniť, podľa potreby používateľa. Z tohto dôvodu sme štandardné kategórie neprogramovali „na tvrdo“, ale spravili sme systém, ktorý vie kategórie vytvoriť podľa súboru ***category.ts.*** V tomto súbore sa nachádza objekt *Categories,* obsahujúci objekty, ktorých názov predstavuje názov kategórie (tento názov sa ďalej pri práci ukladá do položky v tabuľke – *id_category*). Každý objekt ďalej obsahuje dvojicu kľúč-hodnota, predstavujúce názov vlastnosti, ktorý má kategória obsahovať (vlastnosti sú polia ako: name, short_name, place, files, atď.) a hodnotu true, alebo false podľa toho, či chceme aby kategória obsahovala danú vlastnosť, alebo nie. V prípade, že kategória danú vlastnosť nemusí obsahovať, môžeme danú vlastnosť aj úplne vynechať. Aby nedošlo k chybám v programe, používateľ nemôže využívať ľubovoľné vlastnosti, ale len tie, ktoré sú preddefinované. Zoznam všetkých vlastností sa nachádza v komentároch priamo v súbore *category.ts.* Na obrázku vidíme definovanie kategórie Projects :

```
25    Projects:{
26      id: true,
27      id_category: true,
28      name: true,
29      short_name: true,
30      author: true,
31      files: true,
32      date_from: true,
33      date_to: true,
34      note: true,
35      expenses: true,
36    },
37
```

Obrázok 24
Definovanie vlastností kategórie Projects

Ako môžeme vidieť, definovanie kategórií je jednoduché a nevyžaduje žiadne programátorské skúsenosti. Počet kategórií na generovanie nie je nijako obmedzený, takže používateľ môže zadefinovať toľko kategórií, koľko potrebuje. Obmedzením pri tejto objektovej forme definovania je, že používateľ môže definovať kategórie len jednoslovnými názvami, nakoľko nie je možné definovať názvy objektov v typescripte, s použitím medzery. Odporúčame preto, pri nutnosti viacslovných pomenovaní, použiť podtržník. Ďalej odporúčame písať kategórie veľkým začiatočným písmenom a bez

diakritiky, aby sa zachovala jednotnosť názvov, pretože tento názov sa zobrazuje aj v ľavej navigácii.

7.5. Generovanie kategórií

Pri práci s bežnými (generovanými) kategóriami môžeme kód nášho programu rozdeliť do troch celkov. Prvá časť spracováva zobrazenie záznamov jednotlivých kategórií (teda akciu, ktorá sa vykoná pri kliknutí na konkrétnu kategóriu v ľavom menu). Druhá časť je vytvorená pre pridávanie nového, alebo úpravu existujúceho záznamu a posledná časť zabezpečuje zobrazenie detailu konkrétneho záznamu.

Základnou triedou pre zobrazenie záznamov generovaných kategórií je trieda *CategoryComponent.* Celá akcia sa spúšťa po kliknutí na konkrétny odkaz v ľavej navigácii. Následne sa zo zadanej cesty v odkaze vyberie názov kategórie, ktorý sa pošle ako parameter pre servis na back-endovú časť aplikácie. Ako odpoveď z tohto servisu dostávame pole záznamov konkrétnej kategórie. V prípade že daná kategória neexistuje, alebo nastala pri práci s databázou nejaká chyba, vracia servis chybovú hlášku. Chybové hlášky so servisom nie sú zobrazované priamo používateľovi, ale sú vypísané do konzoly. Po naplnení poľa záznamov, sa tento výsledok priamo zobrazuje na prostredie, pomocou súboru *category.component.html.* Aby sme sa vyhli zbytočnému vytváraniu nadbytočných tried, zakomponovali sme do triedy *CategoryComponent* aj funkciu na mazanie jednotlivých záznamov – *deleteItem(id).* Po stlačení tlačidla na zmazanie sa táto funkcia zavolá priamo cez *event listener* s tým, že do parametru sa vkladá id konkrétneho záznamu. Okrem samotného vymazania v databáze, sa v tejto funkcii rieši aj vymazanie záznamu z poľa záznamov, aby sa stránka nemusela načítavať znova a výsledok bol viditeľný hneď po vymazaní.

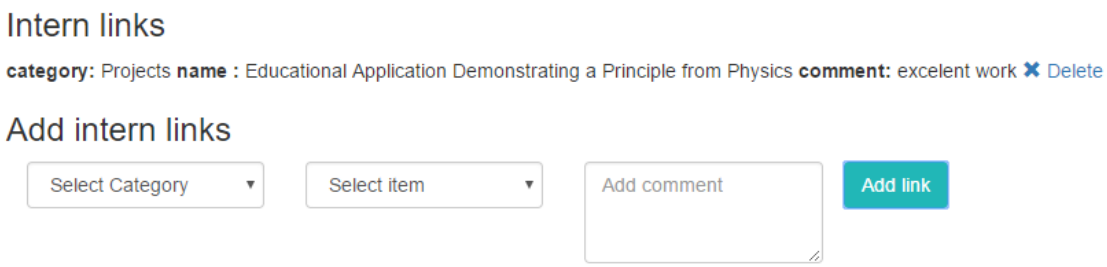
Ďalšou časťou, ktorú sme opisovali, tvorí trieda *NewCategoryComponent,* slúžiaca na vytváranie nových záznamov a editáciu existujúcich. Hneď pri inicializácii triedy je spravené rozdelenie podľa toho, či ide o úpravu, alebo vytvorenie nového záznamu. Toto rozdelenie sa vytvára na základe zadanej adresy vo webovom prehliadači (podľa cesty). Nakoľko v celej aplikácii používame two-way data biding (keď sa zmenia dáta v modeli ,zmenia sa aj dáta vo view a naopak), zmena nastáva okamžite a automaticky, čo zabezpečuje, že model a view sú celý čas aktuálne. Odlišnosť vo vytváraní nového záznamu a úprave je len v tom, že pri vytváraní záznamu sa objekt záznamu len inicializuje a vytvorí prázdny, a pri úprave sa do neho cez servis natiahnu existujúce dáta. Samotný view pre túto triedu tvorí formulár rozdelený na viacero častí, podľa vlastností. Každá vlastnosť, ktorá sa

dá priradiť jednotlivým kategóriám, má vytvorený osobitný kus kódu, ktorý sa zobrazuje len v tom prípade, že je táto vlastnosť nastavená na konkrétnu kategóriu, s ktorou sa aktuálne pracuje. Okrem týchto položiek sa v tejto forme nachádza ešte časť kódu, cez ktorú je možné pridávať externé odkazy a prelinkovania na interné záznamy. Túto funkcionalitu si bližšie charakterizujeme v nasledujúcej kapitole.

Poslednou spomínanou časťou pri generovaných kategóriách bola trieda *DetailCategoryComponent,* slúžiaca na zobrazenie detailu jednotlivých záznamov. Táto trieda je zo spomínaných tried najjednoduchšia. Po kliknutí na detail záznamu, sa z databázy pomocou servisu a http get požiadavky, vytiahne potrebný riadok z tabuľky podľa zadaného ID a ten naparsuje do objektov, ktoré sa vypisuje vo viewe.

7.6. Prelinkovanie záznamov

Jedným z hlavných špecifík aplikácie Pracovná plocha výskumníka je práve to, že je možné prelinkovať na seba ľubovoľné záznamy a tým zabezpečiť jednoduchšiu prácu s údajmi a ľahšie vyhľadávanie potrebných informácií.



Obrázok 25
Prelinkovanie interných záznamov

Funkcionalita na pridávanie linkov v záznamoch je robená v triede *NewCategoryComponent* a funguje nasledovne: používateľ v prvom kroku vyberie v selectboxe ľubovoľnú z existujúcich kategórií. Akonáhle je toto spravené, na pozadí aplikácie sa zavolá servise, ktorý do selectboxu „Select item“ zobrazí názvy všetkých pridaných záznamov z danej kategórie. V nasledujúcom kroku si používateľ vyberie jeden zo záznamov, ktorý chce prelinkovať k práve vytváranému záznamu a pridá komentár, aby aj po čase vedel, v akom vzťahu sú dané záznamy. Pridanie komentára je nutná požiadavka a bez tohto kroku nie je možné pridať dané prelinkovanie. Po pridaní linku, sa informácie

ako id záznamu, id kategórie, meno a komentár, uložia do poľa objektov, ktoré sa neskôr pri ukladaní celého záznamu konvertuje na string a spolu s ostatnými údajmi sa uloží do databázy. Pri editácii je možné prelinkovanie vymazať, čím sa vymažú aj všetky informácie o prelinkovaní aj v databáze.

7.7. Vyhľadávanie

Vyhľadávaniu sme sa venovali aj vo výskume našej práce a tvorí jeden zo základných stavebných kameňov nášho projektu. Vo funkčných požiadavkách sme zadefinovali, že vyhľadávanie musí byť rýchle a efektívne. Vo výskume sme si uviedli niekoľko príkladov, ako fulltextové vyhľadávanie funguje a rozhodli sme sa v našej aplikácii použiť tri konkrétne typy vyhľadávania.

Prvé vyhľadávanie je full-textové vyhľadávanie mena pod konkrétnou kategóriou. Teda hľadajú sa záznamy len vybranej kategórie. Toto vyhľadávanie prebieha priamo na frontende, pomocou knižnice ng2-filter-pipe.

Ďalší spôsob, ktorým vieme vyhľadávať, tentoraz už naprieč celou aplikáciou a to nie len podľa mena, ale aj s použitím iných vlastností, je spravený pomocou štandardnej dokumentácie k vyhľadávaniu v MySql databáze, za pomoci indexových tabuliek. Nakoľko systém sa má využívať v praxi, rozhodli sme sa pre využitie štandardného vyhľadávania, ktoré je overené a efektívne.

Posledným spôsobom vyhľadávania v aplikácii je vyhľadávanie za použitia štítkov. Toto vyhľadávanie sme vytvorili samostatne a funguje nasledovne: pri použití štítku v danom zázname, sa ID záznam s menom zaznamená do zoznamu záznamov pri danom štítku a teda pri samotnom vyhľadávaní sa zobrazí už len zoznam záznamov, čo znamená, že odozva vyhľadávania je veľmi rýchla. Keďže linky na jednotlivé záznamy sú tvorené len za pomoci ID záznamu, nie je nutné robiť pre každý nájdený záznam select z databázy, ale k prelinkovaniu postačuje meno a ID záznam. Nakoľko je štruktúra štítkov stromová, pri zobrazení sa zobrazia aj všetky záznamy v synoch vybraného štítka.

8. Inštalácia a spustenie

Pred spustením aplikácie je nutné nainštalovať na server Node.js, ktorého súčasťou je aj npm. Ďalej je nutné nainštalovať najnovšiu verziu MySQL.

Po týchto krokoch môžeme pristúpiť k inštalácii a nastaveniu samotnej aplikácie. Na priloženom CD máme dva súbory:

- PPV.sql - obsahuje vyexportovanú prázdnu databázu s tabuľkami, ktoré sú potrebné na chod aplikácie.
- PPV.zip – obsahuje adresáre a súbory potrebné pre chod aplikácie.

Zdrojové súbory môžeme taktiež nájsť ako Open-source projekt na webovej adrese: https://github.com/gregi27/PPV1

Postup je teda nasledovný:

1. Naimportujeme súbor PPV.sql do databázy MySQL.
2. Rozbalíme súbor PPV.zip a prekopírujeme všetky položky na server.
3. V súbore „category.js“ nastavíme aktuálne prihlasovacie údaje do databázy.
4. Otvoríme si príkazový riadok a vojdeme do súboru PPV, kde zadáme príkaz „npm install“. Ak inštalácia prebehla úspešne, bez vyhodenia chybových hlásení, pokračujeme v návode, ak nie, ošetríme chybové hlásenia.
5. Otvoríme súbor „category.ts“ a zadefinujeme jednotlivé kategórie.
6. Otvoríme aplikáciu Pracovná plocha výskumníka v prehliadači na adrese nášho servera.

9. Záver

Cieľom diplomovej práce bolo vyvinúť pomerne rozsiahlu aplikáciu, ktorá slúži ako knowledge base a zároveň odkladacia skriňa pre výskumníka a súčasne plní základné funkcie organizéra. Dôležitou súčasťou tejto práce bol aj výskum a následné použitie rýchleho a efektívneho vyhľadávania v danej aplikácii a v neposlednom rade bolo potrebné aj zvolenie vhodného frameworku a programovacieho jazyka na vývoj takejto aplikácie.

Vo výskume sme charakterizovali full-textové vyhľadávanie ako celok, charakterizovali sme podrobnejšie ako full-textové vyhľadávanie funguje a ako sa musia dáta spracovať, aby sa v nich dalo efektívne a rýchlo vyhľadávať. Ďalej sme uviedli praktické príklady z fulltextových vyhľadávaní v relačných databázach MySQL a PostgreSQL a v objektovej databáze MongoDB. Následne vo fáze vývoja sme použili v aplikácii tri typy full-textového vyhľadávania.

V práci bolo porovnaných niekoľko programov, podobajúcich sa svojou funkcionalitou Pracovnej ploche výskumníka, ale dospeli sme k záveru, že je veľmi ťažké nájsť program, ktorý by aspoň z väčšej časti spĺňal základné funkcie a požiadavky vytváranej aplikácie.

Na vývoj bol na základe teoretických východísk použitý jazyk JavaScript a to konkrétne framework Express na serverovú časť aplikácie a Angular 2 na časť klienta. Keďže systém sa má reálne využívať v praxi, pre zabezpečenie responzívneho dizajnu bol použitý ešte ďalší framework a to Bootstrap.

Podarilo sa nám vyvinúť funkčnú aplikáciu, ktorá spĺňa základné požiadavky potrebné pre jej používanie. Medzi základné splnené funkcie patrí: pridávanie záznamov pod kategórie, pridávanie základných vlastností záznamov, pridávanie štítkov, poznámok, externých adries, celkové prelinkovanie jednotlivých záznamov a efektívne filtrovanie v aplikácii. Vytvorili sme základné kategórie záznamov, ktoré je možné jednoducho, bez nutnosti programovania rozširovať. Program je responzívny a nevyhadzuje žiadne chybové hlásenia. Program je uvedený do prevádzky ako open-source a je voľne prístupný na githube pod názvom PPV1.

Webová aplikácia, ktorá je výsledkom tejto diplomovej práce, obsahuje takmer všetku funkcionalitu definovanú v zozname funkčných požiadaviek. Bolo by však vhodné pokračovať vo vývoji aplikácie a rozširovať jej funkcionalitu (napríklad o zálohovanie, zlepšenie dizajnu, upozornenia, zlepšenie funkcie organizéra), aby bola plnohodnotným nástrojom na ukladanie informácií rôzneho typu a organizérom súčasne, čo by v konečnom dôsledku zlepšilo a uľahčilo prácu mnohým výskumníkom či iným používateľom.

Použitá literatúra

[1] Google, Google Drive [cit. 27.11.2016]
Dostupné na: https://www.google.com/drive/

[2] Dropbox, Documentation,[cit. 30.11.2016]
Dostupné na: https://www.dropbox.com/?landing=dbv2

[3] Evernote, Documentation [cit. 30.11.2016]
Dostupné na: https://evernote.com/

[4] Microsoft Windows, OneDrive Documentation [cit. 30.11.2016]
Dostupné na: https://onedrive.live.com/about/en-us/

[5] Katarína Matysová, (2011) Pracovná plocha výskumníka, Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta Matematiky, Fyziky a Informatiky

[6] Tom Hughes-Croucher,(2012). Node: Up and Running. In M. W. Tom Hughes-Croucher, Node: Up and Running. O'Reilly Media Inc.

[7] Anderson, D. (2014). How Node.js Can Accelerate Enterprise Application Development. Modulus.

[8] Nimesh Chhetri, (2016). A Comparative Analysis of Node.js (Server Side JavaScript). ST. Cloud State UNIVERSITY.

[9] Stefan Lutolf (2012). PostgreSQL FULL TEXT Search, Hochschule fur Technik und Wirtschaft University of Applied Sciences

[10] Ashish Trivedi, (2015). Full-Text Search in MongoDB, envatotuts+, [cit. 15.3.2017]
Dostupné na: https://code.tutsplus.com/tutorials/full-text-search-in-mongodb--cms-24835

[11] Lisa Smith, (2016). Indexing for full text search in PostgreSQL, Compose, an IBM Company [cit. 16.3.2017]
Dostupné na: https://www.compose.com/articles/indexing-for-full-text-search-in-postgresql/

[12] Elasticsearch, (2017). Typoes and Mispelings, [cit. 17.3.2017]
Dostupné na: https://www.elastic.co/guide/en/elasticsearch/guide/current/fuzzy-matching.html

[13] Tutorialspoint, (2017). Node.js - Express Framework [cit. 5.4.2017]

Dostupné na:
https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

[14] Rob Eisenberg, (2014). All about Angular 2.0, The Eisenberg Effect [cit. 5.4.2017] Dostupné na: http://eisenbergeffect.bluespire.com/all-about-angular-2-0/

[15] Rauch, G. (2012). Smashing Node.JS JavaScript Everywhere. John Wiley & Sons Inc.

[16] Joseph Delaney, C. G. (n.d.). Node.js at a glance. Whale Path Inc.

[17] Elasticsearch. (2017). Fuzziness. Docs. [cit. 20.3.2017]
Dostupné na:
https://www.elastic.co/guide/en/elasticsearch/guide/current/fuzziness.html

[18] Ryan Levis, (2015). Node web framework comparison [cit. 15.4.2017]
Dostupné na: https://www.pluralsight.com/blog/software-development/node-web-frameworks

[19] Carey Wodehouse, (2016). 15 Frameworks to Know for Next-Level Node.js Development, Upwork Global Inc. [cit. 15.4.2017]
Dostupné na: https://www.upwork.com/hiring/development/15-node-js-frameworks-to-know/

[20] Tutorialspoint, (2017). Spring Framework – overview [cit. 14.4.2017]
Dostupné na: https://www.tutorialspoint.com/spring/spring_overview.htm

[21] Tutorialspoint, (2017). Angular 2 – Architecture [cit. 18.4.2017]
Dostupné na: https://www.tutorialspoint.com/angular2/angular2_architecture.htm