|  |  |  |
| --- | --- | --- |
|  | Универзитет Св. Кирил и Методиј - Скопје  Факултет за информатички науки и компјутерско инженерство | logo_finki |

Дипломска работа на тема:

Plug-in за Microsoft office

**СОДРЖИНА**

Вовед ........................................................................................3

Развојна околина за развивање на plug-in за MS Office ......4

Алатки кои се користат при развивање на апликацијата .....5

Архитектура на апликацијата ...............................................17

Имплементација на апликацијата ........................................19

Дизајн на апликцијата ….......................................................24

Анализа на резултатите што ги дава апликацијата............26

Заклучок .................................................................................31

Користена литература ..........................................................31

# Вовед

Мнозинството на луѓе во светот дел од своето време го посветуваат на компјутер правејќи презентации или пишувајќи некои документи. При започнувањето со пишување на word документ или правење на power point презентација ни се потребни слики за да прикажеме некоја работа што ја опишуваме со текст или да дообјасниме тоа што го пишуваме со текст. Потребните слики ги пребаруваме на интернет. Кога ќе бидеме задоволни со пронајдената слика, сликата најпрво ја симнуваме на тврдиот диск и потоа ја вметнуваме во документот или презентацијата. Половина од времето на пишување на документ или правење на презентација го поминуваме барајќи соодветни слики кои би се вкопиле во нашиот документ. Би се заштедило многу повеќе време доколку постои апликација која го анализира нашиот текст и за поважните зборови ни пронаѓа слики и ние на едноставен начин можеме да ги вметнеме во документот. Како што технологијата се развива побарувањата на луѓето растати можностите за развивање на вакви апликации се поголеми. Подобрувањето на технологијата се прави со цел да му поможи на човекот и да му заштеди време. За таа цел во мојата дипломска работа се потрудив да ја искористам технологијата за човекот да заштеди дел од своето време во пишување на документ или правење на презентации. Plug-in-от (апликацијата) кој го направив како дипломска работа се користи како дел од документот или презентацијат. Со негова помош корисникот може да го обележи текстот за кој сака да пребарува слики, потоа обележаниот текст се анализира со помош на екстерни API-а и добиените резултати се праѓаат на API кое како резултат дава слики. Тие слики можат да се ископираат од plug-in-от и да се вметнат во документот.

# Развојна околина за развивање на plug-in за MS Office

Plug-in за MS Office може да се започне на два начини. Првиот начин е со помош на “NAPA” office 365 развојната околина. Со оваа развојна околина апликациите се развиваат online. За вториот начин како развојна околине се користи Visual studio 2013. За да може да работи со office аппликации потребно е да се инсталира office pack и IIS express 8. Со тие алатки може да се развива апликација offline. Мојата дипломска работа е направена по вториот начин. Microsoft подржува правење на повеќе видови апликации. Тие се:

* Content
* Task pane
* Mail

Така што за различни office програми може да се прават различни апликации. Во продолжение е дадена табела кои office програми какви аппликации подржуваат.

|  |  |
| --- | --- |
| **Application** | **Supported types** |
| Excel 2013 | * Task pane * Content |
| Excel Web App | * Task pane * Content |
| Word 2013 | * Task pane |
| Outlook 2013 | * Mail |
| OWA for Devices | * Mail |
| Outlook Web App | * Mail |
| PowerPoint 2013 | * Task pane |
| Project Professional 2013 | * Task pane |

Мојата апликација е task pane и е подржана само на PowerPoint 2013 и Word 2013. За целосно развивање на апликацијата се користат следните алатки:

* Visual studio 2013
* Office pack 2013
* IIS express 8
* JQuery 1.8.2
* JQuery UI
* AlchemyAPI [1]
* Google image search API v1[2]

# Алатки кои се користат при развивање на апликацијата

Како програмски јазик за развивање на office 2013 апликации се користи javascript за бизнис логиката на кодот, HTML5 за приказот на податоците и CSS3 за дизајн на апликацијата. Microsoft имаат свое javascript API за развивање на апликации[3]. Може да се забележе дека за трите видови на апликации се користи истото API, но во зависност од апликацијата API-то овозможува различни функционалности. Во продолжение ќе бидат прикажани дел од функционалнотите и објектите кои ги нуди API-to.

|  |  |  |
| --- | --- | --- |
| **Object** | **Supported app type** | **Supported host applications** |
| [Appointment](http://msdn.microsoft.com/en-us/library/office/fp160964.aspx) | * Mail app | * Outlook 2013 * Outlook Web App * OWA for Devices |
| [AsyncResult](http://msdn.microsoft.com/en-us/library/office/fp161076.aspx) | * Content app * Mail app * Task pane app | * Excel 2013 * Excel Web App * Outlook 2013 * Outlook Web App * OWA for Devices * PowerPoint 2013 * Project 2013 * Word 2013 |
| [AttachmentDetails](http://msdn.microsoft.com/en-us/library/office/jj984592.aspx) | * Mail app | * Outlook Web App * OWA for Devices |
| [Binding](http://msdn.microsoft.com/en-us/library/office/fp161045.aspx) | * Content app * Task pane app | * Excel 2013 * Excel Web App * Word 2013 |
| [Bindings](http://msdn.microsoft.com/en-us/library/office/fp160966.aspx) | * Content app * Task pane app | * Excel 2013 * Excel Web App * Word 2013 |
| [Contact](http://msdn.microsoft.com/en-us/library/office/fp161005.aspx) | * Mail app | * Outlook 2013 * Outlook Web App * OWA for Devices |
| [Context](http://msdn.microsoft.com/en-us/library/office/fp161104.aspx) | * Content app * Mail app * Task pane app | * Excel 2013 * Excel Web App * Outlook 2013 * Outlook Web App * OWA for Devices * PowerPoint 2013 * Project 2013 * Word 2013 |
| [CustomProperties](http://msdn.microsoft.com/en-us/library/office/fp142138.aspx) | * Mail app | * Outlook 2013 * Outlook Web App * OWA for Devices |
| [CustomXmlNode](http://msdn.microsoft.com/en-us/library/office/fp142260.aspx) | * Task pane app (Word only) | * Word 2013 |
| [CustomXmlPart](http://msdn.microsoft.com/en-us/library/office/fp161160.aspx) | * Task pane app (Word only) | * Word 2013 |
| [CustomXmlParts](http://msdn.microsoft.com/en-us/library/office/fp142202.aspx) | * Task pane app (Word only) | * Word 2013 |
| [CustomXmlPrefixMappings](http://msdn.microsoft.com/en-us/library/office/fp160981.aspx) | * Task pane app (Word only) | * Word 2013 |
| [Document](http://msdn.microsoft.com/en-us/library/office/fp142295.aspx) | * Content app * Task pane app | * Excel 2013 * Excel Web App * PowerPoint 2013 * Project 2013 * Word 2013 |
| [EmailAddressDetails](http://msdn.microsoft.com/en-us/library/office/fp161105.aspx) | * Mail app | * Outlook 2013 * Outlook Web App * OWA for Devices |
| [EmailUser](http://msdn.microsoft.com/en-us/library/office/fp142187.aspx) | * Mail app | * Outlook 2013 * Outlook Web App * OWA for Devices |
| [Entities](http://msdn.microsoft.com/en-us/library/office/fp160984.aspx) | * Mail app | * Outlook 2013 * Outlook Web App * OWA for Devices |

**Jquery[4]** e javascript библиотека koja го олеснува пишувањето на javascript код и полесно се манипулираат DOM елементите од HTML-от. Во мојата апликација оваа библиотека се користи најчесто за додавање на HTML на одредени места. Понатаму ќе биде подетално прикажано користењето на оваа библиотека.

**JQuery UI[5]** е javascript библиотека во која има веќе предефинирани готови компоненти. За да може да се користи JQuery UI мора да е референцирана и JQuery библиотеката. Од JQuery UI во мојата апликација е искористена функционалноста tabs() со која се може да се креираат табови. Понатаму подетално ќе биде прикажано како се користи оваа компонента и некои нејзини модификации.

**Alchemy API[6]** е open source сервис преку кое може да се анализираат текстови. За да може да се користе треба се регистрираме на нивната веб страна и да се прати барање за одобрување на API key за идентификација со кој ќе може идентификува апликацијата. Со тој клуч (key) има ограничени број на повици во еден ден што можат да се прават од една апликација. Резултатите од овај сервис можат да се добијат во сите стандардни формати за разменување на информации помеѓу сервиси, тие се: json, xml, rdf итн. Со помош на ова API можат да се направат следните анализи на текст:

* Entity extraction
* Semantic analysis
* Keyword extraction
* Concept tagging
* Relation extraction
* Text categorization
* Author extraction
* Language detection
* Feed detection
* Text Extraction
* Microformats Parsing
* Content Scraping

Од текстот што ќе се прати на **entity extraction** може да се идентификуваат луѓе, компании, организации, градови, географки поини и други типови на ентитети. На овој повик може да се прати: HTML(за кој се анализира значењето на DOM елементите и текстот што се наоѓа во нив), текст кој сакаме да го анализираме и од кој сакаме да добиеме ентитети кои би ги користеле понатаму, URL(се зема содржината од тоа URL и таа се анализира). Може да се повика со повеќе параметри. Во продолжение се прикажани параметрите со кои може да се повика и нивно објаснување:

* Apikey: привтен клуч кој се земе со регистрирање на страната.
* Text: текстот кој сакаме да го анализираме.
* url: URL до содржината која сакаме да ја анализираме
* outputMode: кој да биде форматот што сакаме да го добиеме како резултат. Подржува xml, json, rdf, rel-tag, rel-tag-raw.
* Jsonp: за callback функција
* Disambiguate: дали да се земаат дополнителни информации
* linkedData: за прикажување на ентитети кои се поврзани со ентитетот кој се врајќа како резултат.
* maxRetrive: бројот на максимален број на ентитети кој може да се дади од еден текст. Предефинирано е 50.

Како резултат со извршување на овај повик можат да се добијат следните резултати:

* Language: го детектира јазикот на текстот
* Type[7]: типот на ентитетот на кој припаѓа даден збор или фраза.
* Relevance: процентот на детекција на зборот или фразата. Вредноста во која се движи е помеѓу 0 и 1.
* Count: колку пати зборот или фразата се наоѓа во текстот што се анализира
* Text: текстот или фразата (ентитетот) што е препознаен
* Disambiguated: дополнителни информации за ентитетот доколку тие постојат. Може да врати географска локација, линк до dbpedia и други веб страни кои даваат дообјаснување за ентитетот.
* Sentiment: може да даде: позитивен, негативен и неутрален.

**Sentiment analysis** е лесен начин да се идентификува дали текстот е со позитивно или негативно мислење. Некои од параметрите со кој се повикува се опишани во entity extraction. Параметрите кои не се опишани во entity extraction ќе бидат опишани тука:

* Apikey
* Text
* url
* outputMode
* jsonp

Како резултат од анализирањето на текстот можат да се добијат следните резултати:

* language
* type: дали е позитивен, негативен или неутрален.
* score: колкав е процентот на сигурност дека текстот е позитивен или негативен.
* mixed: дали во текстот има позитивни и негативни мислења.

Со помош на **keyword extraction** може да се земат поважните теми за кои е напишано во текстот. Тоа се прави со софистицирани статистички алгоритми и со природно процесирање на јазикот (natural language processing), технологии со кој се анализира текстот и се земаат темите. Може да се повикува со следните параметри:

* apikey
* text
* url
* maxRetrieve
* keywordExtractMode: може да биде (normal и strict).
* Sentiment
* outputMode
* jsonp

Со повикување на keyword extraction може да се добијат следните резултати:

* language
* url
* relevance
* text : текстот кој е детектиран

Со помош на **concept tagging** може да се обележат фразите од кој тип се, на која гранка од науката припаѓаат итн. Concept tagging може да се повикува со следните параметри:

* apikey
* text
* url
* maxRetreve
* outputMode
* jsonp
* linkedData

Како резултат од повикување на concept tagging може да се добие:

* language
* url
* relevance: процентот на точност со кој е тагиран зборот или фразата
* text

Со помош на **relation extraction** можат да се најдат сите релации на еден текст. Како релација се смета тројката предмет-акција-објект. Relation extraction може да се повикува со следните параметри:

* apikey
* url
* text
* maxRetreve : предефинирано е 50, максимален број 100.
* outputMode
* jsonp
* sentiment
* entities: дали да се дозволи формирање на релација помеѓу предметот и објектот
* requireEntites: Дали да ги покажува само ентитетите кои формираат релациона тројка
* disambiguate
* linkedData
* coreference: дали да се овозможи прикажување на тој,таа,тоа итн.
* Xpath: xpath прашање со помош на кое ќе може да се филтрираат дел од податоците.

Со повикување на relation extraction може да се добијат следните резултати:

* language
* url
* text: детектираниот предмет, акција и објект од тектот
* sentiment
* verb: дали дадениот збор е глагол, во кое време е, и дали е негиран или не.
* entity: именуваниот ентитет кој е релацијата помеѓу објектот и предметот.

**Text categorization** ја прикажува областа на која припаѓа текстот што сакаме да го анализираме. На пример текстот може да припаѓа на (вести, спорт бизнис итн). Text categorization може да се повикува со следните параметри:

* apikey
* text
* url
* outputMode
* jsonp

Како резултат од повикувањето на text categorization може да се добијат следните резултати:

* category: категоријата на која припаќа содржината која се анализира
* score: процентот на точност за одредената категорија

**Author extraction** се корист за извлекување на авторот на некоја веб страна. Author extraction може да се примени само на содржини кои се HTML па затоа може да се повика само со URL и HTML. Оваа API може дtа се повика со следните параметри:

* html / url: HTML или URL до содржината за која сакаме да го извлечеме авторот.
* Apikey
* outputMode
* jsonp

Како резултат од повикувањето на author extraction може да се добијат следните резултати:

* author: името на авторот на кој припага HTML-от
* url: линкот од кој е пристапен HTML-от

**Language detection** се користи за добивање на јазикот на кој е напишана содржината. Alchemy АPI-то може да препознае повеќе од деведесет и пет (95+) јазици. Language detection може да се повика со следните параметри:

* apikey
* text/html
* url
* outputMode
* jsonp

Како резултат од повикувањето на language detecton може да се добијат следните резултати:

* language
* iso-639-1: стандард за репрезентација на јазици
* iso-639-2: стандард за репрезентација на јазици
* iso-639-3: стандард за репрезентација на јазици
* ethnologue: линк до ethnologue за информации за детектиранит јазик
* native-speakers: број на луѓе што го зборуваат тој јазик
* Wikipedia: линк до Wikipedia за детектираниот јазик

**Feed detection** се користи за извлекување на RSS/ATOM feed линкови од веб страни. Ова API може да се повика само со HTML или URL. Feed detection може да се повика со следните параметри:

* Apikey
* Html
* url
* outputMode
* jsonp

Како резултат од повикувањето на feed detection може да се добијат следните резултати:

* url: линк од кој е повикано API-то
* feed: извлечен RSS/ATOM feed линк.

**Text extraction** се користи за извлекување на текстот од веб страна. При извлекувањето на текстот се прави чистење на HTML страната. Се бришат сите реклами, линкови за навигација и други непотребни содржини. Оваа API може да се повика само со HTML или URL. Text extraction може да се повика со следните параметри:

* apikey
* html
* url
* useMetadata: дали при извлекување да се користи meta-data елементот од HTML
* extractLinks: дали при извлекувањето да се земаат и линковите или не.
* outputMode
* jsonp

Како резултат од повикувањето на text extraction може да се добијат следните резултати:

* url
* text: текстот кој се извлекува од веб страната.

**Microformats parsing** се користи за извлекување на hCard, adr, geo и rel-\* форматите од содржината на страната. hCard е едноставен формат за прикажување на луѓе, компании, организации и места. Geo е формат за прикажување на гео локација, како репрезентација се даваат геогравска широчина и геогравска должина. Adr е формат за прикажување на информации за физичката адреса. Rel-tag е формат за опишување на категории и клучни зборови. Rel-license е формат за прикажување на содржината на специфични лиценци. Microformats parsing може да се повикува со следните параметри:

* apikey
* html
* url
* outputMode
* jsonp

Како резултат од повикувањето на microformats parsing може да се добијат следните резултати:

* url
* microformat: добро структурирани податоци во еден од горенаведените формати.

**Content scraping** се користи за извлекување на податоци од HTML страни со помош на прашања. Alchemy си имаат свој јазик за пишување на прашања (queries). Content scraping може да се повикува со параметрите:

* apikey
* html
* url
* cquery[8]
* outputMode
* jsonp

Како резултат од повикувањето на content scraping може да се добијат следните резултати:

* url
* resultText: структурирани податоци кои се резултат од прашањето(query)
* resultURL: извлечени линкови

Во мојата апликација се користи entity extraction API-то. Во продолжение ќе бидат дадени подетални примери за повикување на ова API и пример на резултат кој го дава тој повик.

**Google Image Search API[9]** се користи за пребарување на слики за дадена фраза или збор. Ова API е направено од google и ги дава истите резултати како што ги дава и google.com во табот images. Сервисот е компактибилен со скоро сите верзии на веб пребарувачи. Пребарувачите кои го подржуваат google image search се: firefox 1.5+, safari, IE6+, opera 9+ и Chrome. Google image search може да се вметне директно во javascript кодот и да се користи внатре со помош на google објектот. Исто така ова API има подршка и за други развојни околини Flash, Java, PHP, Python и Perl. За да може да се избегне cross domain може да се користат и callback функции. Во мојата апликација od google image search API-то се користи само GET повици кои како резултат даваат JSON со позната структура. Сликите во JSON се дадени како линкови до оригиналните слики и заради тоа многу лесно можат да се прикажат во HTML како src атрибут од img елемент. При повикувањето на оваа API има само два задолжителни параметри, сите други се опционални параметри. Двата задолжителни параметри се:

* q: каде што се вметнува фразата или зборот кој сакаме да се пребарува. Зборот или фразата мора да е декодира со помош на фунцкијата encodeURIComponent() пред да се вметни за пребарување.
* v: каде што се вметнува верзијата. Во моментот google image search API подржува само 1.0 верзија.

Останатите опционални се:

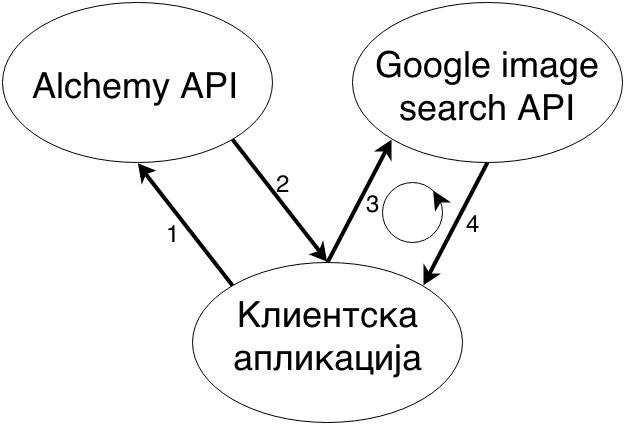
* as\_filetype: со овој параметар се кажува сликите од кој формат да бидат. Подржани се JPG, PNG, GIF и BMP форматите. Доколку не е наведен овој параметар ќе се пребаруваат сите формати.
* as\_rights: со овој параметар може да се земат сликите кои имаат одредени права. Можни параметри се cc\_publicdomain, cc\_attribute, cc\_sharelike, cc\_sharealike, cc\_noncommercial, cc\_nonderived.
* as\_sitesearch: може да се ограничи пребарувањето само на специфилен домеин.
* callback: Доколку сакаме да ни се повика некоја функција во javascript кодот која би се извршила одкако ќе се добие резултатот. Исто така се користи за избегнување на cross domain забрани.
* context: Овој параметар е поврзан со callback. Се кажува во функцијата која ке се повика на кој context припаѓа.
* hl: овој параметар служи за јазикот кој се користи на локацијата од каде што е пратено барањето. Предефиниран е англиски јазик
* imgc: овој параметар кажува дали да се бараат само црно-бели слики или слики во боја. Можни параметри се gray и color
* imgcolor: Овој е параметар е експериментален, што значи дека резултатите што ќе ги врати не се веродостојни. Се користи за кажување на бојата која ќе преовладува во сликата. Можни параметри се: black, blue, brown, gray, green, orange, pink, purple, red, teal, white, yellow.
* imgsz: Со овој параметар може да се каже големината на сликите кои сакаме да ги добиеме како резултат. Можни параметри се: icon, small, medium, large, xlarge, xxlarge, huge.
* imgtype: Овој параметар е исто така експериментален. Се користи за да се земат сликите што се појавува одредена форма на нив. Можни параметри се: face, photo, clipart, lineart.
* rsz: Овој параметар се користи за да се каже колку резултати може да се земат со еден повик. Вредноста што може да му се зададе е цел број помеѓу 1 и 8
* safe: Овај параметар се користи за ниво на безбедност. Може да биде едно од следните параметри: active, moderate, off
* start: Овој параметар се користи за од која слика да почнуваат да се прикажуваат. Се користи заедно со параметарот rsz за да можат да се земат повеќе од 8 слики.
* userip: Овој параметар се користи за да се испрати интернет адресата на корисникот до API-то.

При добивањето на резултати може да се земат максимум 64 слики со 8 барања. Како резултат при повикувањето на ова API се добиваат следните резултати:

* cursor: кој е поврзан со start параметарот. Во него се прикажани од каде ќе треба да се почне за да се земат следните податоци, која е тековната страна на слики што се зема, кој е линкот за добивање на повеќе резултати.
* results[]: Тоа е поле од од JSON објекти кои содржат информации за сликата. Информациите што ги содржи се:
  + content: информации за сликата
  + contentNoFormatting: поврзано со content, но е дадено заедно со html елементите
  + height: висина на сликата во пиксели
  + html: го дава целото DOM стебло заедно од родителот на елементот каде што се наоѓа сликата па се до елементот кој ја содржи сликата.
  + originalContextUrl: На кој линк се наоѓа сликата.
  + tbHeight: висина на сликичката во помали размери (thumbnail)
  + tbUrl: линк до сликичката во помали размери
  + tbWidth: ширина на сликичката во помали размери
  + title: наслов на сликата
  + titleNoFormatting: наслов на сликата заедно со HTML елементот каде што е пронајдена
  + unescapedUrl: линк до сликата каде што не се декодирани сите букви
  + url: линк до сликата
  + width: ширина на сликата.

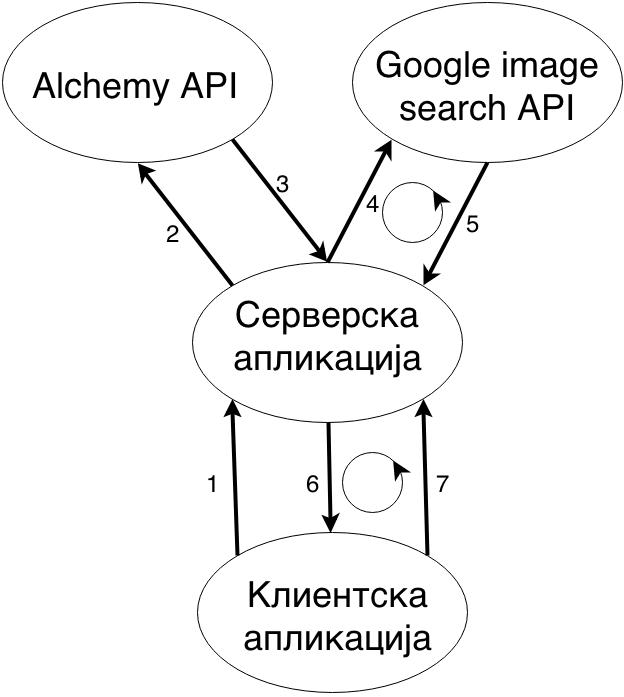
# Архитектура на апликацијата

Генерално апликацијата се содржи од 3 слоеви. Еден слој кој се извршува на страната на клиентот, а другите два слоеви се двете API-а кои ги повикува клиентцката страна. На следната слика е прикажана архитектурата и чекорите кои се превземаат



Може да се забележи дека одкако ќе се анализира текстот со alchemy API потоа се земаат сликите парциално. За секоја фраза или збор од анализираните зборови се прават по 8 повици од google image search API. Тие 8 повици се прават синхроно (еден после друг). Повиците за пребарување на други зборови се прават кога корисникот ќе го одбери табот во кој се наоѓа зборот или фразата. Со одбирањето на табот, зборот од табот се праќа до google image search API и повторно се прават нови 8 синхрони повици.

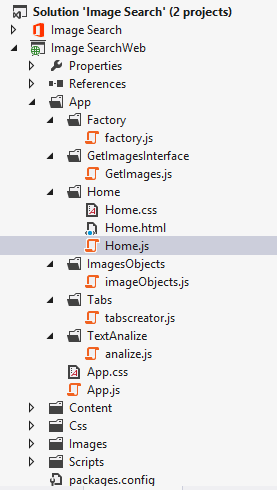
Подобрување би се направило доколку апликацијата би пристапувала до серверска страна, а серверската страна ќе пристапува до двете API-a. Во тој случај доколку се изврши промена на било кое API корисникот нема да ја почуствува таја промена. Позитивната страна на тоа е што и кодот на клиентска страна си останува ист се менува само кодот што се наоѓа на серверската страна. Во таков случај може да се направи и самостоен алгоритам кој ќе ги одвојува зборовите или композиција од API-a од каде што би се подобрило пребарувањето на сликите за даден текст. На следната слика е прикажана таквата архитектура.



Од сликата може да се забележи дека клиетската апликација прави комуникација само со еден систем. Во првиот чекор се праќа текстот. Серверската страна ги прави повиците до двата сервиси и потоа на клиентот му дава резултат и линкови за да ги земи следните слики. Потоа серверот продолжува да праќа повици до сервисите и резултатите ги зачувува локално. Притоа клиентот кога ќе побара слики за некои зборови доколку веќе ги има на серверската страна тогаш се даваат ќе бидат пратени до клиетската страна, ако ги нема тогаш повторно се прави повик до сервисите и резултато се проследува до клиентот. Со оваа постапка би се забрзало работењето на апликацијата заради тоа што се праќаат помалку повици, но во овој случај серверот станува „тесно грло’’ и ќе треба да опслужува повеќе апликации одеднаш. Но, ист така и ќе има повеќе безпотребни повици до сервисите. Што значи во зависност од ресурсите кои ги имаме на располагање можеме да избираме кое решение би било подобро за проблемот кој сакаме да го решиме. Доколку ќе треба да се надградува по често во тој случај второто решение е подобар избор. Доколку надградувањето ќе биде во многу ретки интервали тогаш веќе имплементираното решение е подобар избор. Самата клиетска апликација е поделена во два делови. Првиот дел е конекторот кои ги поврзува апликацијата со Microsoft office, додека вториот дел е веб делот. Тоа е како обична веб апликација која може да се виде преку веб пребарувач. Microsoft office за прикажување на ваквите апликации користи Internet explorer со тоа што сите негови недостатоци ги наследнуваат сите Microsoft office апликации. Кога се стартува апликацијата најпрво се покренува IIS express сервер каде што се става апликацијата. Потоа конекторот ги поврзува заедно со Microsoft office. Во продолжение е дадена целосната имплементација на апликацијата.

# Имплементација на апликацијата

Како што е погоре напоменато апликацијата на клиентска страна се состои од два делови. Делот што ги поврзува апликацијата и Microsoft office е xml датотека. Во оваа датотека е опишано од кој хост ќе се зема апликацијата, кој е креатор, кој продукт од Microsoft office ќе треба да се стартова за користење на апликацијата. Додека апликациски дел е HTML, javascript и CSS3. Мојата апликацијата има само една HTML страница и на неа динамички се додаваат табовите заедно со сликите. Исто така, има и една CSS датотека која служи за дизајнот на дадениот HTML. Javascript-тите како што беше кажано во архитектура на апликацијата се поделени во 3 главни групи. Првата група се грижи за додавањето на DOM елементите на постоечкиот HTML. Втората група се користи за повикување на сервисот за анализирање на текстот и третата група се користи за добивање на сликите од сервисот за слики. На следната се прикажани сите скрпти кои се користат во апликацијата:



Во скрипта factory.js се наоѓаат сите функции кои не се поврзани со трите главни делови. Заради cross-domain забрани сите повици кои се прават до сервисите ќе треба да бидат како дел од нов елемент script кој се додава во html-от. Factory.js содржи функција која додава и брише script елемент од HTML-от. Исто така во оваа скрипта се наоѓаат функции за вметнување на слика во HTML-от. Заради тоа што аликацијата може да работи на word и на power point вметнувањето на сликите во двете програми се прави на различен начин. Тука е направена апстракција на повикот кон функцијата за вметнување на слики, така што, тоа да се прави само со еден повик и самата скрипта знае да го изврши правилниот код во зависност дали се работи за word или power point. GetImages.js се користи за превземање на сите слики од google image search сервисот. Сликите се сместуваат во локални променливи и потоа се исцртуваат. Доколку сакаме да го замениме сервисот со друг сервис ќе треба да се замени повикот и новиот резултата да се конвертира на начин за да може да се зачува во локалните објекти. Еден пример за повикување на google image search сервисот е следниот:

<https://ajax.googleapis.com/ajax/services/search/images?v=1.0&rsz=8&start=0&q=cars&callback=service.processResults>

Овој повик како резултат го дава следниот JSON:

service.processResults({

* **responseData**: {
  + **results**:[
    - {
      * **GsearchResultClass**: "GimageSearch",
      * **width**: "1280",
      * **height**: "960",
      * **imageId**: "ANd9GcSwjL-uvKeWqYljLYD9ox5dQAkUeGSg9QBgmTj9JdB\_8upFnbJDUjVjGDEr",
      * **tbWidth**: "150",
      * **tbHeight**: "113",
      * **unescapedUrl**: "<http://images2.fanpop.com/image/photos/13800000/farrari-sports-cars-13821367-1280-960.jpg>",
      * **url**: "<http://images2.fanpop.com/image/photos/13800000/farrari-sports-cars-13821367-1280-960.jpg>",
      * **visibleUrl**: "www.fanpop.com",
      * **title**: "farrari - Sports <b>Cars</b> Wallpaper (13821367) - Fanpop fanclubs",
      * **titleNoFormatting**: "farrari - Sports Cars Wallpaper (13821367) - Fanpop fanclubs",
      * **originalContextUrl**: "<http://www.fanpop.com/clubs/sports-cars/images/13821367/title/farrari-wallpaper>",
      * **content**: "Sports <b>Cars</b> farrari",
      * **contentNoFormatting**: "Sports Cars farrari",
      * **tbUrl**: "<http://t2.gstatic.com/images?q=tbn:ANd9GcSwjL-uvKeWqYljLYD9ox5dQAkUeGSg9QBgmTj9JdB_8upFnbJDUjVjGDEr>"

},],

* + **cursor**: {
    - **resultCount**: "6,100,000,000",
    - **pages**: [
      * {
        + **start**: "0",
        + **label**: 1

}, ...],

* + - **estimatedResultCount**: "6100000000",
    - **currentPageIndex**: 0,
    - **moreResultsUrl**: "<http://www.google.com/images?oe=utf8&ie=utf8&source=uds&start=0&hl=en&q=cars>",
    - **searchResultTime**: "0.27"

}},

* **responseDetails**: null,
* **responseStatus**: 200})

Може да се забележе дека не се прикажани сите резултати заради тоа што има големо повторување на резултатите. Заради оптимизација и побрзо работење на апликацијата, повиците кон сервисот се прават кога на корисникот ги пребарува сликите од даден збор или фраза. Со тоа што доколку корисникот побара слики за два зборови истовремено тогаш се чека прво за заврше повикот за првиот збор па потоа и за вториот збор.

Во home.js се наоѓаат сите функции кои се повикуваат кога ќе се притисне некое од копчината. По притискањето на некое од копчината се повикуваат останатите скрипти за да може да се заврше саканата операција. imageObjects.js ги чува сите слики кои се прикажуваат на корисникот. Некои од овие зачувани слики се внесуваат во документот. Со помош на GetImages.js се превземаат сликите и се зачувуваат во објекти во imageObjects.js. tabCreator.js се користи за прикажување на табовите во апликацијата. Табовите се поделени во два реда. Во првиот ред се прикажани категориите на кои припаѓаат зборовите, додека во вториот дел се прикажани зборовите и сликите кои се наоѓаат во тој таб. Исто така корисникот може и самиот да си пребарува слики по зборови или фрази. Добиените слики од пребарувањето се појавуваат во посебен таб и исто така можат да се ставаат во документот. Analize.js се грижи за анализирање на текстот. Со текстот кој е селектиран се прави повик до следниот линк: <https://access.alchemyapi.com/calls/text/TextGetRankedNamedEntities?jsonp=analizator.analizeText&apikey=a788ae67c0770b3eb652c6433b1286894a4a7c69&outputMode=json&text>= каде што потоа следува текстот. Со овај повик се добива следниот JSON:

analizator.analizeText(

{

* **status**: "OK",
* **usage**: "By accessing AlchemyAPI or using information generated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use: http://www.alchemyapi.com/company/terms.html",
* **url**: "",
* **language**: "english",
* **entities**: [
  + {
    - **type**: "Organization",
    - **relevance**: "0.900867",
    - **count**: "2",
    - **text**: "Faculty of Electrical Engineering",
    - **disambiguated**:

{

* + - * **name**: "University of Belgrade Faculty of Electrical Engineering",
      * **dbpedia**: "<http://dbpedia.org/resource/University_of_Belgrade_Faculty_of_Electrical_Engineering>",
      * **freebase**: "<http://rdf.freebase.com/ns/m.0bwkq1b>"

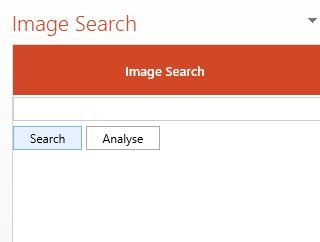
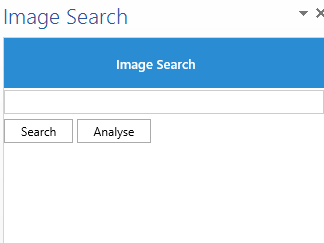
}...},

]})

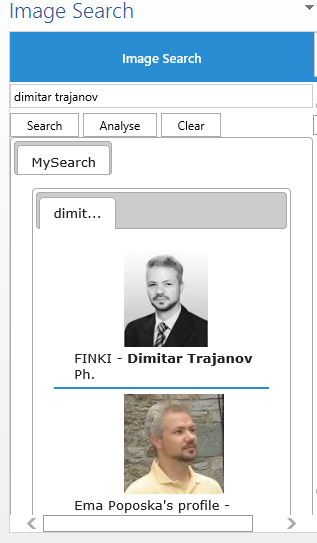
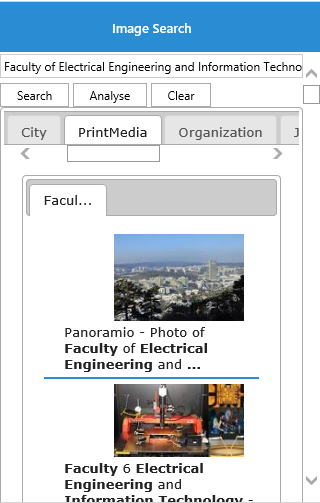
Може да се забележи дека исто како и сервисот за слики и овој сервис се добива со помош на callback функција. Доколку статусот е “OK” тогаш се продолжува со земање на зборовите од полето. Доколку статусот е различен од OK се прави нов повик, доколку и за тој повик статусот е различен од OK на корисникот му се прикажува порака дека не е успеано да се добијат резултати за текстот. Добиените JSON се изминува и секој член со групира според атрибутот type. Сите елементи на кои им е ист атрибутот type се ставаат во еден таб. Доколку фразите се долги тогаш на табовите се прикажува дел од фразите, а додека цела фраза може да се виде во делот за пребарување. App.js се користи за прикажување на било каква порака на корисникот.

# Дизајн на апликцијата

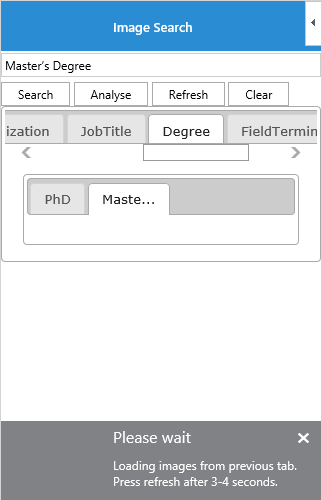
Заради малиот простор за приказ на апликацијата, апликацијата има доста едноставен дизајн. На сликата е покажан почетниот дизајн за work (лево), power point(десно)



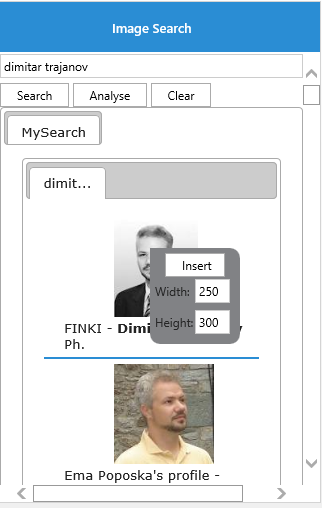
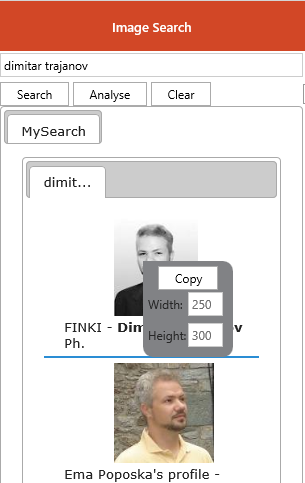
Дизајнот за word и power point е речиси целосно идентичен. Деловите кои се различни ќе бидат прикажани посебно. Копчетно Search се користи доколку корисникот сака да пребарува слики за некоја фраза или збор. Доколку има прикажано табови со слики се појавува копче Clear кое се користи за да може да се врати апликацијата во првобитната состојба (да има само две коплина и сите слики и табови да се избришани). На следната слика (лево)е прикажано пребарување на слики за одредена личност. Насловот на сликите исто така се добива од сервисот за слики (JSON-от е прикажан во погорната тема)

Доколку е зададен некој текст и се анализира тогаш се појавуваат повеќе табови, така што, првиот на почетокот се земаат само слики за првиот таб. На горната слика (десно) е прикажан изгледот на апликацијата во дадената состојба. Доколку се избере прво еден таб потоа друг таб тогаш можно е да се појави грешка затоа што сликите од првиот таб сеуште не се превземени од сервисот. Во тој случај се појавува refresh копче за да може да ги превземеме повторно сликите. Тоа е прикажано на следната слика:



За вметнување на слика во документот или презентацијата доволно е да се кликне на сликата. После кликнувањето излегува прозорец кој е различен во однос на тоа дали се работи за документ или презентација. Вметнувањето на слика во документ се прави со копчето insert и се внесува големината на сликата што корисникот сака да ја внесе. За презентација корисникот притиска на копчето copy и потоа прави paste во презентацијата со тоа што се внесува сликата со оригиналната големина. На следните две слики е прикажано додавање на слика во документ (лево) и во презентација (десно).

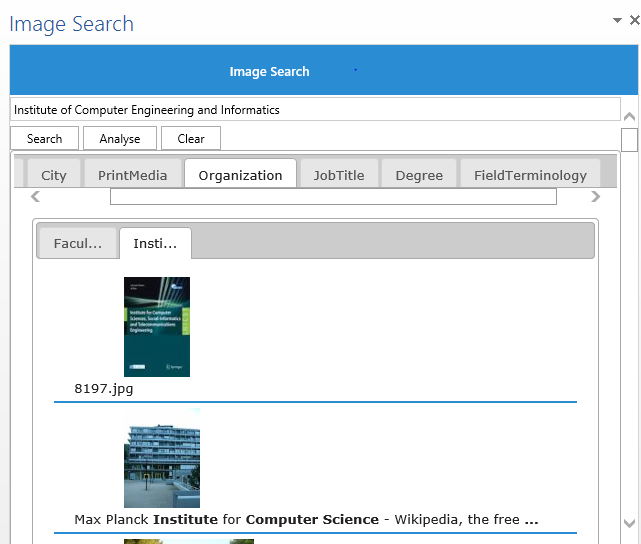
 

# Анализа на резултатите што ги дава апликацијата

За анализа на резултатите од апликацијата ќе го анализираме следниот текст:

„Born in 1972 in Radovish. Got his Bachelor’s and Master’s Degree at the Faculty of Electrical Engineering in Skopje, in 1996 and 1998, respectively. Got his PhD at the Faculty of Electrical Engineering and Information Technologies in Skopje in 2006. He’s been working as a junior teaching assistant at the Institute of Computer Engineering and Informatics at the Faculty of Electrical Engineering in Skopje since 1999, and in 2001 he was elected as a teaching assistant. Right after he got his PhD in 2006, he was elected as an assistant professor, and in 2010 he was elected as an associate professor. As an assistant professor and an associate professor, he’s been teaching numerous courses in the fields of web-based systems, wireless computer networks, public mobile networks, network programming, system software, electronic and mobile commerce, software fault tolerance and computer architectures. He has a lot of experience in holding professional trainings, has received instructor certificates for numerous industrial courses and has held trainings for companies in Macedonia. He’s published more than 70 scientific papers on international and domestic conferences, and has co-authored 4 books. He has managed and been part of numerous international and domestic scientific research projects. He’s also managed more than 15 application development projects. He’s a creator of a web-based teaching and studying supports system, which is used by the staff and students of the Faculty of Electrical Engineering and Information Technologies in Skopje. His fields of scientific research are: semantic web, wireless ad hoc networks, complex networks, web applications and services and system-on-ship design.’’

Во текстот е дадена кратка биографија за деканот на ФИНКИ д-р Димитар Трајанов. Може да се забележи дека во текстот се јавуваат главни фрази како град во кој е роден, град во кој студирал, на кои универзитетит завршил и со кои титули, кои предмети ги предава итн. Сите тие поважни зборови се пронаоѓаат и се пребаруваат. Но при пребарувањето не се дава никаква поврзаност со текстот. На пример: Доколку се пребарува за градот радовиш тогаш пребарувањето се врши за градот радовиш. На тоа пребарување не му се дава до значење дека текстот е за Димитар Трајанов и тоа да се поврзе со градот. На следната слика се прикажани категориите и зборовите кои се пронајдени од апликацијата за овај текст.



Категориите и зборовите заедно со прицентот на точност за тие зборови што можат да се добијат од анализирање на текстот се:

* City
  + Radovish – 28.2692%
  + Skopje – 81.1811%
* PrintMedia
  + Faculty of Electrical Engineering and Information Technologies – 70.5425%
* Organization
  + Faculty of Electrical Engineering – 66.9159%
  + Institute of Computer Engineering and Informatics – 38.1452%
* JobTitle
  + assistant professor – 54.0575%
  + teaching assistant – 46.2656%
  + associate professor – 44.1135%
  + instructor – 21.869%
* Degree
  + PhD – 34.3476%
  + Master’s Degree – 27.2133%
* FieldTerminology
  + semantic web – 22.9977%
  + fault tolerance – 22.1023%
  + web applications – 21.834%
  + mobile commerce – 21.0582%
  + system software – 20.7014%

Дел од резултатите што се прикажани за градот Скопје се следните:

Може да се забележе дека сликите што се прикажани се резултатите што се прикажани за град Скопје се прилично добри и дека повеќето од сликите можат да се искористат. Да ги разгледаме резултатите што се дадени за „Faculty of Electrical Engineering and Information Technologies’’:

Од сликите може да се забележе дека тие слики се за факултетит низ целиосвет, но во текстот се споменува за факултетот во скопје. Доколку на пребарувањето се додаде фразата „in Skopje‘‘ тогаш резултатите би биле многу по точни и повеќе би можеле да се искористат за текстот е даден. Новите резултати се прикажани на следните слики:

  http://t3.gstatic.com/images?q=tbn:ANd9GcQi-GXKvUygJ8CIdG4HtO0S9jZmIQKGTKCZBSH4upk1p9np6VGxYaP5Fn4c 

Кога се прошири текстот за пребарување може да се забележи дека сликите што се даваат како резултат се повеќе прецизни за дадениот текст одколку доколку се пребарува без „in skopje’’. Заради не прецизноста на некој од резултатите корисникот може да си ги дополнува фразите кои се пребарани и да ја пребара новата фраза. Со тоа би добил резултати кои се повеќе прецизни за текстот кој се анализира, а со тоа и повеќе искористливи од страна на клиентот. Следен текст кој ќе го разгледаме ќе биде „assistant professor’’ од категоријата JobTitle. Може да се забележат дека овие фрази се премногу општи и пребарувањето на слики за тие фрази ќе даде резултати кои многу малку можеме да ги поврземе со текстот. Резултатите што се добиени од пребарувањето на оваа фраза се:

Исто така може да се забележи и дека процентот дека овие зборови се важни во текстот се движи помеѓу 50%, што е прилично голем процент, но сепак сликите што се даваат се скоро бескорисни. Доколку би се продолжил фразата која се пребарува со зборот „in skopje’’ или „at faculty of electrical engineering and information technologies in skopje’’ резултатите кои се пребаруваат би се подобриле. Резултатите добиени од пребарувањето на „assistant professor in skopje’’ се:

Додека резултатите од пребарување на фразата „ assistant professor at faculty of electrical engineering and information technologies In skopje’’ се:

Што се забележува дека многу се поврзани со текстот кој се анализира одколку погорните резултати каде што се пребарува само таа фраза. Што може да се доведе до заклучок дека за добивање на подобри резултати ќе треба да се дополнуваат фразите или да му зададе контекст за кој се пребарува. Контекстот за пребарувањето исто така може да се излече од текстот со различни методи, но се забележува и дека за некои зборови не е потребно додавање на нови фрази. Заради тоа дали на зборовите би се додало некоја фраза за подобрување најповеќе зависи од текстот, категоријата на која припаѓа тој збор.

# Заклучок

Можностите кои ги нудат технологиите на денешницата се огромни, прашањето е дали сме спремни и способни да ги искористиме и да создадеме продукти кои би го олесниле животот и би му заштедиле време на човекот. Апликации за Microsoft office може да се пронајдат на маркетот на Microsoft, но бројот на апликации не е премногу голем. Креирањето на мојата апликација е направено со HTML кој е служи за прикаживање на податоци на веб уште од почетоците на користење на вебот. Заради тоа многу многу лесни програмерите можат да се ориентираат на оваа технологија и брзо да направат апликација. Се разбира, апликацијата ќе треба да е за корисниците што значи исто како и секоја друга апликација или веб страна не треба да е премногу комплицирана, но побитно е кога ќе се прикажува некоја идеја на корисниците да биде меѓу првите на пазарот. Заради тоа, корисниците се навикнуваат на апликациите што се прво појавени и се навикнуваат на нивниот изглед. Според моето истражување ваква апликација за office сеуште не постои што се надевам дека би можело да се пронајди во најповеќе спуштаните апликации на тој маркет.

# Користена литература

[1] <http://www.alchemyapi.com/>

[2] <https://developers.google.com/image-search/>

[3] <http://msdn.microsoft.com/en-us/library/office/fp142185.aspx>

[4] <http://api.jquery.com/>

[5] <http://api.jqueryui.com/>

[6] <http://www.alchemyapi.com/api/>

[7] <http://www.alchemyapi.com/api/entity/types.html>

[8] <http://www.alchemyapi.com/api/scrape/qlang.html>

[9] <https://developers.google.com/image-search/>