Cognitive Assisted Blogger

Bachelor's Thesis

By

Oliver Sjöstrand and Maximillian Vilensten.

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

In this thesis a cognitive application is developed using IBMs cloud platform Bluemix and its Watson services. The goal of the application is to make it easier for bloggers and journalists to publish online during conferences. To reach the goal the application was designed with, among other things, functionalities for speech to text, publish on social media and automatically generated hashtags.

The project was initially focused on developing a Bluemix hosted Node.js web application that would work on both iOS and Android devices. After constructing the Minimum Viable Product (MVP) for the web application the focus of the project changed and instead APIs were developed for the purpose to aid the development of native iOS and Android applications. The APIs were developed as Node.js applications with the help of API Connect and was afterwards hosted on Bluemix. This consisted of three separate APIs; the first API creates hashtags based on location and also tries to figure out who is speaking, to do this Facebook Graph and Alchemy was used. The second API makes it possible to save and get text from a cloud database and utilizes Cloudant. Lastly, the third API saves audio and also returns transcribed text and hashtags with the help of Object Storage, Watson Speech to Text and Alchemy.

Keywords: Cognitive, Speech to Text, Cloud, API, IBM, social media

Sammanfattning

I förevarande examesarbete är en kognitiv applikation utvecklad med hjälp av IBMs molnplattform Bluemix. Målet med applikationen är att göra det lättare för bloggare och journalister att publicera under konferenser. För att nå målet designades applikationen med funktioner för bland annat tal till text, uppladdning till sociala medier och automatiskt genererade hashtags.

Inledningsvis var projektet inriktat på att utveckla en Node.js webbapplikation som skulle fungera på både iOS och Android enheter. Efter att ha konstruerat en Minimum Viable Product (MVP) för webbapplikationen ändrades projektets fokus och istället skapades API:er i syfte att underlätta utvecklingen av dedikerade iOS och Android applikationer. API:erna utvecklades som Node.js applikationer med hjälp av API Connect och kördes sedan på Bluemix. Detta bestod av tre separata API:er; första API:et skapar hashtags baserat på plats samt försöker lista ut vem det är som talar, för att göra detta används Facebook Graph och Alchemy. Det andra API:et gör det möjligt att spara och hämta text i molnet och använder sig utav CloudantDB. Avslutningsvis det tredje API:et sparar ljud och även returnerar transkriberad text och hashtags med hjälp utav Object Storage, Watson Speech to text samt Alchemy.

Nyckelord: Kognitiv, Tal till text, Molntjänster, API, IBM, sociala medier

Acknowledgments

First off we would like to thank Brandon Jones for starting the project and allowing us to do this thesis on IBM. Anne-Bell Larsson at IBM also deserves a thank you for being an important advisor to us.

Thank you to our supervisor Mats Lilja and to our examiner Christian Nyberg for the guidance.

The iOS and Android developers Heber Andrade together with Klas Zetterlund who we been working close with needs special thank you for developing the mobile application as well as providing ideas and input.

Finally we would like to thank all the other developers at IBM who have given us guidance during the project.

Oliver Sjöstrand and Maximillian Vilensten

Contents

Preface

This thesis report is divided in to three phases. Phase one was dedicated to the planning of the product. Phase two is the development of the product as a Web application which was the initial goal of this thesis. Phase three is the development of three different APIs to aid the development of native iOS application and native Android application.

Just as the MVP version of the Web application was ready IBM gave us a new task. Two App developer students from Malmö Högskola joined in on the project and their assignment was to build native Android and iOS applications of the same project as specified in phase one. Our task was now instead shifted in to building backend APIs to support their applications. Therefore a third phase was added which was dedicated to the planning and development of the APIs.

1. Introduction

1.1. Background

IBM is one of the world leading companies in information processing. With more than 400 000 employees and thousands of business partners across the globe they try to give their customers the combined competence of their entire worldwide company, one of the steps to achieve that goal was for IBM to invest in multiple IT solutions that focuses on the cloud and Internet of things such as their IBM Bluemix and Watson.

IBM strives to be among the top three cloud service providers in the world, for this to be possible they need a wide range of software to utilize their services so that it can be recognized as a strong alternative for developers. In order to maintain a top three spot the service also needs to be continuously upgraded so that new and exciting features that makes it stand apart from other services will be added.

IBM gave us the task to develop a show case for the utilization of IBM Bluemix and IBM Watson in a consumer application. After discussions with IBM about what software we should develop both parties agreed on an application that should help bloggers and journalists in their work and to make them “heard”.

1.2. Goal

The minimum viable product goal is to develop a web application that will have an audio recording feature where the user can, at any time, press a button to analyze a speech. The app will with the help of IBM Watson's Speech to Text application return a text string that can easily be edited and after that shared to Facebook and Twitter with the push of a button. Before the user shares the content he will get suggestions on suitable hashtags. The app will also have a history feature that will allow the user to see statistics over the content shared on social media through the app. When the application is done it be evaluated and analyzed to see if the goal was reached or if improvements need to be done to Bluemix and Watson.

1.3. Problem Specification

The following questions are the core problems of this thesis:

- Does IBM Watson's speech to text analysis (see chapter 2.3) work reliably enough to implement the system?
- Will Alchemy API (see chapter 2.5) be efficient enough to filter out names from Text?
- Can speeches in text form effectively be stored and received from the cloud?
- Is the API Connect service (see chapter 2.4) mature enough to make the process of creating secure APIs simple?

These questions are fundamental for this thesis since the product relies on these features.

A pleasing user experience cannot be guaranteed if for example Watson's Speech to Text service cannot provide accurate results.

1.4. Limitations

The project will be limited to using IBM Bluemix platform and therefore all the cognitive aspects of the project will be implemented with the use of IBM Watson services. The different APIs in this thesis will be implemented with the help of API Connect for added security.

1.5. Purpose

The competition for bloggers and journalists today is tough, especially with the number of internet blogs constantly growing. One way to get an edge over the competition is to be the first one to publish content from for example a press conference. This can be very time consuming and stressful since the blogger/journalists needs to keep track of what the speaker says and at the same time take notes and update their media channel.

The purpose with this project is to reduce the time taken for the user to update their media channel and make it easier to share quotes from a speaker. The hypothesis is that this will allow the user to update their channel more frequently and the likelihood of missing something that the speaker said is reduced.

2. Technical Background

This chapter gives a short introduction to the different software's and technologies that are used during this project.

2.1. Bluemix

Bluemix is a new cloud platform that IBM provides, it is an implementation of IBM's open Cloud Architecture which is based on Cloud Foundry open technology. Cloud foundry is a platform as a service (PaaS) and allows Bluemix to handle many of the major coding languages such as Java, Node.js and Ruby. The idea behind Bluemix is to have applications constantly running in the cloud so that they can be available all the time, examples of such applications are APIs and storing data in the cloud.

2.2. Watson

Watson is a computer system developed by IBM specifically to analyze and answer natural language questions on Jeopardy. Watson competed in 2011 on Jeopardy and won the first price of one million USD which was donated to charity. In 2013 it was announced that Watson would be used for utilization management decisions in lung cancer which was also the first time the computer would be used in commercial purposes. In 2014 IBM added multiple Watson services to IBMs cloud platform Bluemix which meant that anyone now could access Watson's cognitive capabilities. For more information see [1].

2.3. Speech to Text

The Watson Speech to Text (STT) service is provided through IBM's Bluemix platform. As the name suggests the service takes speech that the user sends through a REST API either as an audio stream or as a complete sound file. The audio will go through a series of mathematical analyses that will guess which words the user said and compare them with a word database to see which real word was closest and replaces the assumed word with this.

There are many cases where this is not enough, for example when words have similar pronunciations but different spellings. Examples would be sole and soul or sent and scent. To solve this Watson needs to compare the entire sentence with a database containing information on grammar and different schematics on sentences. After this the words are ready to be sent back to the user. The data that the user will get from the Speech to Text API will be sent in the JSON format. Figure 1 shows an example from the IBM website.


```
{
   "results": [
      . . .
      "keywords_result": {
         "Chuck": [ {
               "normalized_text":"Chuck",
               "start_time":0.76,
               "confidence":0.984,
               "end_time":1.18,
            }],
         "100": [
            {
               "normalized_text":"one hundred",
               "start_time":5.87,
               "confidence":1.0,
               "end_time":6.64,
            },{
               "normalized_text":"one hundred",
               "start_time":11.03,
               "confidence":1.0,
               "end_time":11.81,
            }
         ]}
      . . .
   ]}
```


Fig. 1. Example of JSON-data

As seen in figure 1 a filtering feature is used which reacts to the two words Chuck and one hundred (100). When the user triggers the keyword Chuck the API sends back a JSON object with the normalized text, the start and end time of when the speaker used the word, and finally a word confidence score which is how confident the API was that this actually was the word spoken. The API also recognizes the number 100 as the normalizer text one hundred and vice versa which makes it easier to filter numbers.

In this project the Speech to text will be using the normalized_text and confidence values.

The idea of Speech to Text has existed for a long time and the process described in this section is based on an array of different publications that IBM found to work best for their Speech to text service. See appendices 10.1 for publications.

2.4. IBM API Connect

API Connect is a new software from IBM released in March 2016 that allows its users to create, run, manage, and secure APIs and Microservices. API Connect uses a built-in gateway to integrate StrongLoop and IBM API Management. For more information see [2].

2.4.1. StrongLoop

StrongLoop is a company bought by IBM in September 2015 and is one of the leaders of the Node.js community. StrongLoop offer developer tools for creating Node.js applications with API and CURD capabilities. The application can be configured to run on-premises or in the cloud.

The Loop Back Node.js framework is an open source framework made by StrongLoop to build APIs from scratch. LoopBack helps developers to make APIs that can access data from various databases like MongoDB, SOAP, and REST APIs. LoopBack is often used in a combination with StrongLoop Arc which includes a graphical tool to edit, deploy, and monitor LoopBack apps. StrongLoop Arc is now also combined with IBM API Management and a part of IBM API Connect. For more information see [3] and [4].

2.4.2. IBM API Management

IBM API Management allows users to create new APIs by defining existing proxies or assemble new APIs by mapping together backend systems. Users can develop and update existing versions of their APIs without any service disruption. When the API is created the user can manage access and choose their own authentication method like HTTP, OAuth 2.0 or authentication through an URL. Having an API protected with an authentication allows the API Admin to control who gets access, and how many requests that is allowed in a given time space. API Management also helps developers to set up a developer portal where people can subscribe and access the created API. For more information see [5].

2.5. AlchemyAPI and Entity extraction

AlchemyAPI is a company that uses similar technology as IBM Watson for machine learning to do natural language processing like semantic text analysis and sentiment analysis. AlchemyAPI was bought by IBM in March 2015 to accelerate their development of cognitive computing capabilities and was at the same time added as a Watson service to IBM Bluemix.

AlchemyAPI offers multiple APIs with cognitive capabilities. A few examples are: Sentiment analysis, keyword extraction, author extraction and, language detection. They also provide image analysis APIs like Face Detection and Image Tagging but in this project their entity extraction API will be used. For more information see [6].

When the user sends a text string or URL to a website with text to the API via a HTTP request the API will try to detect words that fall in to different categories. The API will then in JSON return the word and what category it falls in to.

Entity extraction API is capable of identifying categories like people, companies, organizations, cities and movies from a text. Altogether entity extraction is capable of extracting close to a thousand different entity types. For more information see [7].

2.6. Facebook Graph API

The Graph API allows developers to read and write data into Facebook and presents a view over the social graph where each node and edge represents an object in the graph (e.g., people, photos, events and pages).

An example of how a request can look like:

https://graph.facebook.com/v2.5/57843015021?fields=events&access_token=REPLACE_WITH_TOKEN

This query will return information about events occurring at the O2 arena including event description, event start time and event end time.

The number 57843015021 is a unique id for a location, in this case it's the O2 arena in London. The parameter fields specifies the information that will be returned, in this case event information. Token is an ID for the app or user that makes the request to the Graph API, the token can be required at www.developers.facebook.com.

See appendices 10.2 for an example of how the response can look like. For more information see [8].

2.7. Hashtags

The use of hashtags was introduced on Twitter as a way to mark the tweet with a specific topic and allow other users to find the tweet while searching for specific content. Hashtags consists of a string of characters preceded by the pound symbol (#). This combination acts like a label for the message itself. If a user clicks on the hashtag it acts like a search function to find all other tweets with that same label. A tweet with a hashtag can look like this:

"how do you feel about using # (pound) for groups, As in #barcamp [msg]?"

(This is credited as the first Twitter hashtag sent August 2007 by Chris Messina) For more information see [9].

The hashtag function on Twitter has since been embraced by the community and is activity used to find other tweets and people with the same interests.

The hashtags have spread to not only social media sites like Facebook, Instagram, Google+, and Pinterest but also offline in in the linguistic landscape, for example advertising, headlines, and political slogans.

Even if hashtags was born on Twitter as a way to categorize a method, it is now used in many different ways and purposes. See appendices for more examples of how hashtags are used in social media messages. For more information see [9] .

2.8. Node.js

Node.js, also called Node, is a runtime environment that is open source, cross-platform and made for developing server-side Web applications. Many of Nodes basic modules are written in JavaScript and it is also often the language used when developers write new ones. It is based on Google's V8 engine and focused on performance and low memory consumption. Node.js uses an asynchronous I/O evening model, meaning that it does not rely on multithreading to support concurrent execution like most other environments. This approach in combination with the JavaScript language makes it easy to create asynchronous functions that can be registered as event handlers. For more information see [10].

2.8.1. Ionic

Ionic is a Node.js module that is an open source mobile app framework for developing native applications for devices that run Android, iOS, Windows 10, Blackberry OS, and more. Ionic is built on top of Angular and Apache Cordova. Applications made with Ionic are programmed in JavaScript, HTML and, CSS. For more information see [11].

2.8.2. Express

Express is a server-side Node.js framework that provides features for building web applications. Express defines a routing table which is used to preform different actions based on HTTP method and URL. It also allows the user to set up middleware to respond to HTTP Requests and dynamically render HTML pages based on passing arguments to templates. [12].

3. Method

3.1. Project model

For this project the group decided to work with the Kanban model with restrictions to the IBM garage work method. This was a unanimous decision made by all members of the group and also the company (IBM) after a discussion on how to handle the work process in this project.

The initial work method is Kanban with a unique set of rules in order to keep all parties involved as informed as possible.

The work method starts with the group having a design meeting, at the start of this meeting a key issue that the group should work to fix. In our case it could have been something like "how can Watson's speech to text improve my everyday life?" or something along those lines. After this the group decides a persona in order to decide what features that should be developed and which of these that should be in the Minimum Viable Product (MVP) and which features that should be in the backlog. It is also in this stage where user stories, tasks and the prioritization of all the tasks is being made.

After the initial prioritization and documentation step the group can start working on the highest prioritized tasks and work its way down. In order to keep track on which person is doing what and how the development is going a Kanban board named Trello is being used. In this application the customer can re prioritize, add new tasks, see what has been done, and also see which group member is doing what. The application can also be used by the developers in order to see what has to be worked on and what has been done already.

The group has weekly playbacks which is a form of standup meetings where all members show what they have produced to the customer, and also tell what they have planned to do the following week. This is so that the customer have a chance to get a better overview of the project and also have the ability to prioritize on the fly but it's also an opportunity for the group to have a say if they will not be able to produce something in time etc.

Because of the weekly meetings the sprints stretch over a one week period and the planning/sprint ending meetings are being held after the weekly meeting with the customer.

3.2. Source criticism

In order to ensure that the Thesis holds an unprejudiced view and the ability for the reader to rely on the information in the document, a rather strict method of checking and finding sources was used. When searching for information it was important to first and foremost search for articles and publications that are accepted by the university such as LUBsearch or the Publications and dissertations section on the lth website. This is quite time consuming and the results from these can be very interesting and give depth and understanding to different subjects, they are usually not specific enough to use as sources on specific services from different companies such as IBM bluemix. So in many cases books or internet pages were also or in some cases exclusively used to find information.

For the sake of Source criticism books are a very good tool since the facts probably already have been checked by the author and/or the publisher of the book, but when searching on the web this is not always true. And because of that a more critical view is needed when finding facts there. To make sure that the webpage entered can be considered "reliable" it has to pass a series of guidelines:

1. How is the webpage edited?

Some pages such as ne.se are edited by people that have knowledge in the subject while some pages such as Wikipedia can be edited by anyone and can therefore be incorrect for longer periods of time before being corrected

2. Who owns the website (what is in it for the author)?

Corporations tend to glorify technologies they use or are in possession of in order to get more sales and can therefore be slightly unsuitable to use in some cases. For example in cases where the website states what makes their product different from the competition.

.

3. Does the webpage give a serious impression?

There are many institutions and groups that give information that provides a good and objective source of information while some tend to give a more subjective and less serious impression which usually is unsuitable for thesis work.

4. Does the webpage have sources?

Even though a source seems serious and contains a lot of well written information on it the information can come from bad sources. Therefore a check is made to see if there are sources and if these sources seem reliable.

5. Is this the correct domain?

In some cases there are webpages that have similar domain names but one is not from the original author and is created by other people to trick the reader or webpages that claim to be from the original author but is not. These cases are uncommon but still a check is required to make sure that this is the correct webpage. For more information see [13].

3.3. Individual grading of source credibility

The sources numbered [1] to [7] are websites that IBM provide for documentation of the different Watson services. The sources would have been deemed not reliable if they would have been used to compare the products against competitors since companies have tendencies of glorifying their products. In this case they are only used to figure out how the services work and to get information about the technical background. The webpage is also updated frequently which is why these sources are deemed highly trustworthy.

The sources numbered [8],[9],[13],[14] and [16] are all articles written by people from different universities and can be found in LUBsearch meaning the articles must have been deemed good enough for the universities to let these articles to be published. With that in mind and while also passing this thesis procedure to find credibility in articles these sources are deemed highly trustworthy.

The sources numbered [10] and [11] are webpages from companies and while these pages update frequently not all information is updated. But they are deemed quite trustworthy for the purpose to get general information about the different services as in this thesis.

Source [12] is directly from the Swedish National Agency for Education and is updated every time a change that concerns Education is made. This source is deemed highly trustworthy.

Source [15] is a website made to help people with html5 the website does not have a serious name and does not seem to have any quality check before uploading any information. This source is deemed not trustworthy and therefore the code examples taken from the webpage to the thesis were double checked to make sure that they work before being put in.

4. Phase one - Defining the product

4.1. Introduction

The goal with phase one is to define the target audience of the application and with that information also define the key features of the product. A decision will also be made on what platform the application should be available on. The phase should result in a more specific project plan and give enough knowledge of the project so that an estimation can be made on what features will be possible to develop in the given time space of the thesis.

4.2. The given assignment

IBM presented a predefined problem which they wanted to solve with an application. The problem was a scenario where a person attends a conference and listens to a speaker. After the conference the person wants to tweet a specific quote from the speaker but he or she cannot remember exactly what the speaker said.

Instead of having to guess what the speaker said or trying to search on internet to find the quote, it should be possible to record it with a speech to text application that also allows the user to share the quote easily and fast.

4.3. Target audience

The application is mainly targeted towards bloggers and people who attend conferences, wants to stay "connected", and often share content on social sites media like Facebook and Twitter. However considering that the application will be solving the problem with a speech to text function and options to share that text, anyone who wants to speak their own tweet instead of typing may want to use the app. This way of using the application could potentially be very useful for people with functional impairment who has difficulties to type on a small phone keyboard.

4.4. Personas

While developing a product it is important to understand the end users' needs and what he or she may want out of the product which in itself can be a challenge since users often do not know what they want. Personas is a technique that was introduced by Alan Cooper and that is commonly used while designing a product. For more information see [14].

Personas are fictional characters made up to represent users that might use the product. The reason to create personas is to help designers to go back to a specific persona with its own story and ask for example the question "Would Anna need this feature?" instead of "Would the user need this feature". The term user is often too wide and abstract and makes it therefore harder to make decisions with the user in mind.

In this project multiple personas was created but a persona named Anna was mostly used and was made to represent the key users of the application.

4.4.1. Persona Anna 27

Anna is 27 years old woman who has a blog and a twitter account where she makes posts about the latest tech. Anna strives to get more followers so that she will be able to earn enough money to support herself on blogging. Anna's strategy to get more followers is to be the first one out with new tech information, therefore she often goes to tech conventions where she posts live to her blog and twitter. A problem that Anna has while she is at conventions is that she gets stressed having to type and listen to a speaker at the same time. She is worried that she will miss or forget something when she does multiple things at the same time.

4.5. Application features

A list of possible application features was made by using the personas technique and trying to answer the following question

"What app features could help Anna be less stressed out when she wants to post during a convention?"

4.5.1. Speech to Text

To record speech from the users device and transcribe it to text in real-time and display the text on the screen.

4.5.2. Post text to Facebook and Twitter

This would allow Anna to edit and post the text transcribed by the speech to text function directly to Facebook and twitter without having to copy and paste into another app.

In future versions this feature will be implemented to support as many social media sites as possible.

4.5.3. Add a picture to the post

This function would allow Anna to take a picture during the convention and post it together with the transcribed text to her Facebook and Twitter accounts.

4.5.4. Save speech text

A button or automatic save function to save the transcribed speech text locally on the device. Anna can then at any time view the speech text from conventions she has been at.

4.5.5. Save speech audio

If Anna wants, she can choose to save the recorded speech as an audio file so that she can listen to the speech again.

4.5.6. Display the speakers name

The app will automatically try to find out and then display the speaker's name.

4.5.7. History and Statistics

A page on the app where Anna can read her latest posts and some statistics for example when it was posted, how many likes/retweets it has and who the speaker was.

4.5.8. Get tweet notification

Anna can choose to get notifications from the app when someone is using the same generated hashtags as her or when someone has retweeted her posted speech.

4.5.9. Generate hashtags

Gives Anna relevant hashtag recommendations based on the speaker or event she is at when posting to Facebook and Twitter.

4.5.10. Generate hashtags from a picture

When Anna takes a picture to upload to Facebook and Twitter the app will automatically generate and give recommendations on hashtags based on the content in the picture.

Following things was also taken in to consideration while making decisions about the functionality:

- The app should be easy to use.
- No unnecessary features.
- The user should not have to create an account.
- It has to work seamlessly.

It was also determined that it had to be an app that that was easy to use and that could run on a mobile phone. This way Anna only needs to bring the phone to the convention.

4.6. Minimum Viable Product

Minimum Viable Product (MVP) is a version of a product which is complete enough to demonstrate the value it brings to its users. An MVP requires less time to be completed compared to final product and should only include its most essential elements, even if the MVP version is missing planned functionality that the users soon will be requesting anyway.

When a MVP is complete it is put through tests to confirm or refute its value and growth hypotheses. This way a company can measure the products traction on the market and save resources on development in case a product would not fulfill the hypotheses. For more information see [15].

In this project the MVP features was chosen based on the hypothesis stated in project purpose. Figure 2 shows what features will be implemented in the MVP.

App features	MVP version	Later version
Speech to Text	X	
Post to Facebook and Twitter	X	
Add a picture to the post		X
Save speech audio		X
Display the speakers name	X	
Save speech text	X	
History and statistics		X
Get tweet notification		X
Generate hashtags	X	
Generate hashtags from a picture		X

Fig. 2. List of application features.

4.7. Choosing platform and development tools

IBM wished that the developed app should be possible to run on both Android and iOS devices without having to write dedicated code for each device. This made sense since the app does not target a specific user group that only uses one platform such as IPhones.

There are several services that provide development for both Android and iOS such as React Native and Adobe PhoneGap. Since the group had little experience with these programs the choice stood instead between two possible solutions of making the same code runnable on both Android and iOS devices.

First solution was to use the Ionic framework to build a hybrid app. The framework then takes the code and compiles it to native Android and iOS applications

Second solution was to make a web app that can run on a phone's internet browser.

A few small test projects was made using both Ionic and Node.js to evaluate the programs and to see which one would be the best for developing the app.

A problem with Ionic was that there was no information available about how to integrate different Bluemix services with Ionic/Cordova using an SDK. Even if Ionic and Cordova has a fairly large community it still did not have much documentation regarding how to use the platform with Watson's speech to text service.

Node.js on the other hand had an SDK to help integrate different Bluemix services and also more documentation about how to develop Node.js web applications with Bluemix. IBM even had a demo speech to text site on https://speech-to-text-demo.mybluemix.net/ that is powered by Node.js and with the source code available for anyone to download.

Because of the lack of experience developing in JavaScript it was important to have as much documentations and code examples as possible. Therefore it was decided that a Node.js web application would be developed.

4.8. Result of phase one

One of the goals for this phase was to determine what functionality the application should have. The decisions was made during a design workshop at IBM. This way a discussion could be held about each and every function of the application and prioritize them in a way that IBM also was contended with. Together with IBM the following prioritization list was made as seen in figure 3:

App features	MVP version	Later version
Speech to Text	X	
Post to Facebook and Twitter	X	
Save speech text	X	
Generate hashtags	X	
Display the speakers name	X	
Save speech audio		X
History and statistics		X
Add a picture to the post		X
Get tweet notification		X
Generate hashtags from a picture		X

Fig. 3. Prioritized list of the application features.

It was decided that the application would be made as a Web app hosted on a Node.js server. The decision was made based on the fact that there is considerably more documentation and a working SDK for Node.js projects. The Node.js application could then also easily be uploaded to Bluemix and hosted there.

5. Phase two - Web application

5.1. Goal

The goal with phase two is to both plan and implement the functions of the MVP as stated in 4.6 as a Node.js web application.

5.2. Planning the Web application

5.2.1. Design

The express framework for Node.js was selected to be used for hosting the Node.js webserver which sends the projects index.ejs file to the client. The file then generates the user interface for the application.

Since only a MVP version of the product would be developed in this phase the interface was decided to be as simple as possible with just a basic layout and buttons for the different functionalities. The only design criteria for the MVP is that it should be a responsive web design. In later versions more focus would be made on design and making it a better user experience.

A working interface was coded in JavaScript, HTML and CSS as a prototype. The design of this can be seen in figure 4. The functionality for each button could then be added afterwards.

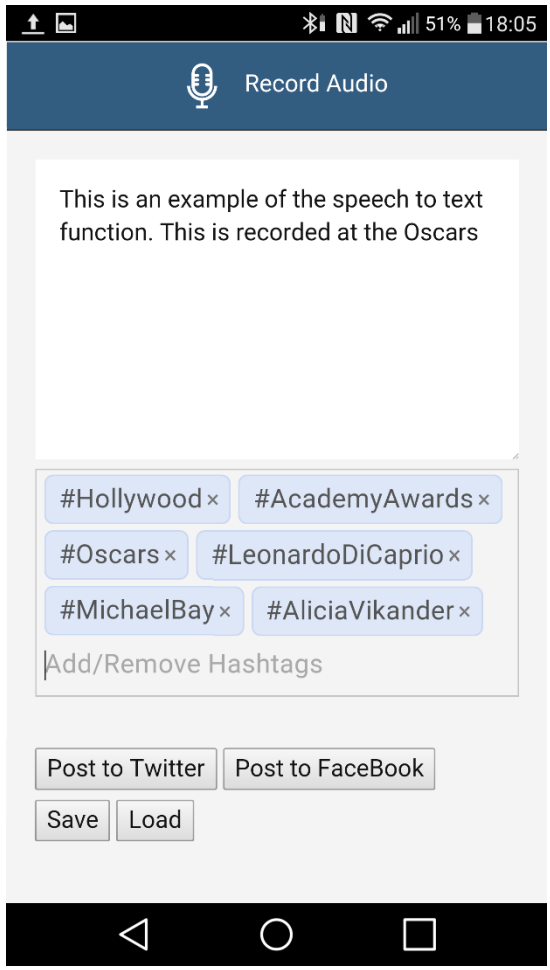

Fig. 4. Prototype of the Web application.

5.2.2. Speech to Text

The speech to text function was chosen to be implemented with IBM's own Watson speech to text API. This was both a desire from IBM but also a preferred choice since the API is already included as a Bluemix service. Watson speech to text service is also the speech to text software with most documentation when it's used in combination with Bluemix.

5.2.3. Post to Twitter and Facebook

Posting to Facebook and Twitter could either be done client side (the device running the web application) or server side (the Node.js server hosting the web application).

Doing it server side would mean that the web application uses OAuth 2 to first ask the user for permission to access their Facebook or Twitter account. If the user grants the application access a token will be received from the API and sent to server side. The server then has access to post messages using either the Facebook or Twitter API.

Posting messages client side on the other hand can be done in multiple ways and would mean that the Facebook and Twitter SDKs could be used to make the implementation easier. The only drawback is that the Twitter API don't support image upload from the client without running an Oauth.

It was decided to keep it as simple as possible and develop it client side since the goal of the phase was just to make an MVP version of the product.

5.2.4. Hashtag recommendations

It was decided that a stand-alone REST API hosted as a Node.js application that generates hashtags would be a good solution. The web application can then do a simple HTTP request and the API will send a response with the hashtags. This way the API can be used for other applications as well and the code that generates the hashtags can easily be modified without affecting the web application as long as the output is in the same format.

Different solutions was discussed regarding how the API would generate relevant hashtags. It was concluded that the device should get longitude and latitude coordinates that is then included in the query for the API request. The API could then take the coordinates and use them to try and find out what event the user is at with the help of other APIs.

A third party API named Ticnet API was considered since they offer information about events that can be found based on location. Using their API would have a few drawbacks that needed to be accounted for. One big drawback was that Ticnet API would only work for the Nordic countries, and only to events which they have sold tickets to. There is also not much documentation and support available since the API is not that widely used.

It was concluded that using the Facebook Graph API would most likely be sufficient to give relevant hashtag recommendations. With the Facebook Graph API Facebook-registered events can be found based on the coordinates and the event description can then be sent to Alchemy Entity Extraction API for analysis. The extracted words are then hopefully related to the event that the user currently is at and can be used as hashtags. If the Facebook event description contains the name of the speaker the Alchemy API will be able to extract it as well. This way the Hashtag API would also allow the Web app to display the speakers name but as a hashtag (see 4.5.6).

Several tests where made to see what results could be expected from the Hashtag API. Using the example description as showed in appendices 11.3.1 from the Science Museum in London and sending it to the Alchemy API returns an output (appendices 11.4.1) with the categories Person, Facility, Organization, FieldTerminology, JobTitle and Quantity. In this example, by using the city name, venue name, and then extracting values from the categories Person and Organization the following hashtags was received:

#London, #ScienceMuseum, #BuzzAldrin, #BrianCox, #NASA, #NeilArmstrong, #SchoolofPhysicsandAstronomy, #Universityofmanchester

Figure 2 was made to show the structure of the Web application.

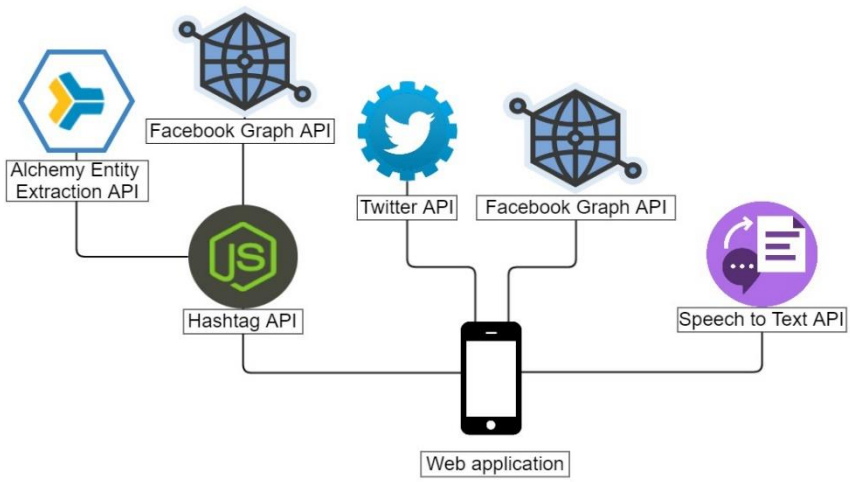

Fig. 5. Diagram of the Web application

5.3. Implementing the functionalities

5.3.1. Speech to text

IBM's demo speech to text application allows users to record audio with a microphone, the transcribed text and keywords will then be displayed in real time on the screen. It is also possible to upload an existing audio file that will then be transcribed with the result displayed on the screen.

The demo project was used as a template for the application in this thesis since it offered the real-time transcription functionality that was wanted. This was done to save time so that other functionalities of the web application could be prioritized.

When the Node.js application server is launched a Bluemix speech to text service username and password will be sent to the Watson API with the help of the Vcap environment variable. A response will be received with a generated speech to text token, this token is used to authenticate the application when the WebSocket to Watson STT is initiated.

After the token is received the server will wait for a client to make a HTTP request to the website. If a client makes a connection the server will send an index.ejs file back to the client who will execute it.

The coordinates of the device will be gathered right away when the Web application is loaded and sent to the Hashtag API. The hashtags will then be displayed in the application if they were successfully received.

When the user press the record button a WebSocket between the client and Watson speech to text API is opened. To open the WebSocket the client will first send an initial request to a URL with the token and language setting as a query parameter

```
wss://stream.watsonplatform.net/speech-to-text/api/v1/recognize?watson-token=TOKEN&model=en-US_BroadbandModel
```

TOKEN in the URL is the generated token that the Node.js server received when it was started. If the token is valid Watson STT will accept the connection and return a connection confirmation to the client. As soon as the connection is set up the client will send an "action request" which tells the STT service what to do. In this case the action "start" will be sent. A confirmation {"state":"listening"} is sent back to the client and the microphone on the device running the web application is activated. The audio captured by the microphone is sent in blobs through the WebSocket to the server. The API then transcribes the audio and returns the generated text. The WebSocket connection will stay open to send data back and forth until the client closes it by pressing stop recording.

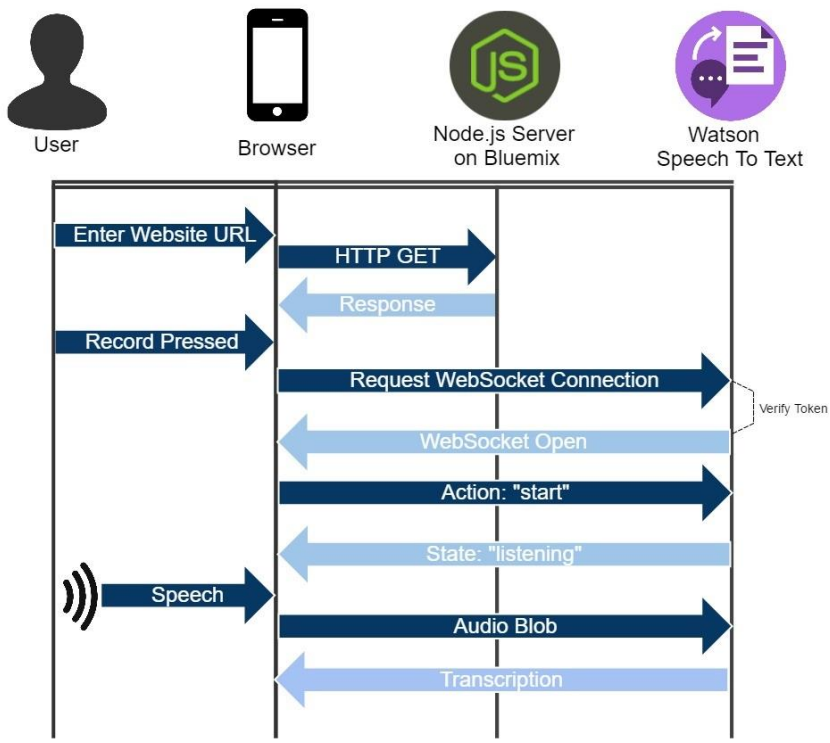

Fig. 6. Web application with Speech to Text.

Watson STT returns interim JSON results in response to the call. For each interim result, the final field of the JSON response is set to false; it is set to true for the final result. This is done because Watson STT takes the words context in account when transcribing, so a transcribed word may change based on the words spoken afterwards. This allows the API to return the transcribed text in real time instead of returning one sentence at a time.

5.3.2. Post to Facebook and Twitter

Posting to Facebook and Twitter is actually two separate functionalities and tasks since they use separate APIs. Therefore granting the application access to the accounts had to be handled separately.

TWITTER

Twitter offers a invoke URL function which made the implementation incredibly simple. The invoke function allows developers and users to simply paste in a string in the query of the URL and when the request is made the user will be directed to a twitters webpage with the query string already inputted in a post to the "post tweet" field. In addition to that, twitter also provides web intents for the invoke URL that adjusts its format depending on what device it's running on.

Using this method the posting to twitter functionality was easily implemented without having to register an app on Twitter that then has to ask for permission by the user.

FACEBOOK

Unfortunately Facebook does not offer the same simple solution as Twitter. Instead the Facebook SDK for JavaScript was used to handle Facebook login and also the permission requests to the user.

When a user loads the web application the app will check if the user is logged in to Facebook, if the user is not logged in a "Log in" button will be displayed instead of "Post to Facebook". Unlike Twitter, the first time the user wants to make a post to Facebook a pop-up window is displayed where the user is asked to give the application access to post to his Facebook wall.

5.3.3. Web SQL storage

Initially it was planned to use the local storage feature that comes as a standard in HTML 5. The reason why local storage was chosen is due to the high capability with browsers and with devices such as android and iOS which it a very favorable option. Local storage can also contain 5mb of data which is plenty enough for text and also much larger than the 4 kb available from cookies. It was however quickly discovered that while using express java the data saved with local storage was only saved within the temporary location on the client's computer and was therefore deleted when the client ended the session.

In order to solve this problem an attempt to simulate local storage on the server side was implemented instead. This was a very bad solution since this used space that the server needed on Bluemix making the homepage slow or even unstable when multiple users where on the homepage at the same time.

After two failed attempts to create a local save feature using local storage a decision to drop local storage and to research other solutions to the problem, and this solution would be WebSQL.

Web SQL is a service that can be used directly in HTML5, it uses SQL queries to create a database on the client's local storage. To do this a variable needs to be implemented that defines the name, which version of WebSQL that should be used, a description of the database and finally the size, an example of how this looks in JavaScript code would be:

```
var db = openDatabase('mydb', '1.0', 'my database ',25
* 1024 * 1024);
```

In order to later use the database specific functions are used. To simplify this in the thesis an example of how to use the function to create a database and then insert a text into the database would be good:

```
db.transaction(function (tx) {
  tx.executeSql('CREATE TABLE IF NOT EXISTS Speech
(id unique, text)');
  tx.executeSql('INSERT INTO Speech (id, text)
VALUES (1, "My new saved text ")');
});
```

To receive values form the database the select command from standard SQL can still be used and is a fairly simple thing to use, an example to retrieve text and write it out would be like this:

```
tx.executeSql('SELECT * FROM Speech', [], function
(tx, results) {
  var len = results.rows.length, i;
  for (i = 0; i < len; i++) {
    alert(results.rows.item(i).text);
  }
});
```

In order to be able to save the five last texts and still be able to work on devices with a low amount of storage a decision was made to make the database 25 mb large. Where every text then could be 5mb large which 5000000 characters large texts and will probably not be too small for any speeches. For more information see [16]

5.3.4. Hashtag API

The first thing the Hashtag API needs to do after receiving longitude and latitude coordinates through its query is to use those coordinates to find all surrounding Facebook events. This turned out to be tricky since it is not possible to simply receive event information based on coordinates with the Facebooks API. The easiest solution would have been to use FQL which is similar to SQL. This would allow the application to use a FQL like this:

```
'SELECT  name, venue, location, start_time, eid
FROM event WHERE eid IN (SELECT eid FROM
event_member WHERE uid IN (SELECT uid2 FROM friend
WHERE uid1 = me()) AND start_time > '. created_time
.' OR uid = me()) AND start_time > '. created_time
.' AND venue.longitude < \''. (long+offset) .'\'
AND venue.latitude < \''. (lat+offset) .'\' AND
venue.longitude > \''. (long-offset) .'\' AND
venue.latitude > \''. (lat-offset) .'\' ORDER BY
start_time ASC '. limit;
```

Unfortunately FQL has been deprecated although it still worked fine at the time of this thesis. The newer Facebook Graph was used instead.

Using Facebook Graph instead of FQL to get events is not as elegant since two queries to the Facebook API is needed.

The first query sends a request to receive all the Facebook locations in an area around the coordinates. A response will then be received with all the location IDs within the area.

```
"https://graph.facebook.com/v2.5/search?type=place&
q=*&center=" + req.query.lat + "," + req.query.lng
+ "&distance=" + req.query.distance +
"&limit=1000&fields=id&access_token=" +
req.query.access_token;
```

The parameter distance is the radian in meters of the area from where the locations will be fetched. The distance variable is just like the coordinates set in the query when a call is made to the Hashtag API. Access_token is the access token to Facebooks API and is not needed in the query since the app has one hard coded.

The second query then takes all the received location IDs and checks each one to see if it has a registered event.

```
"https://graph.facebook.com/v2.5/?ids=" +
idArray.join(",") +
"&fields=id,name,cover.fields(id,source)" +
",location,events.fields(id,name,cover.fields(id,so
urce),description,start_time,end_time,attending_cou
nt,declined_count,maybe_count,noreply_count)"
+".since(" + (currentTimestamp - 604800) +
").until(" + (new Date().getTime() / 1000 +
3600).toFixed() + ")&access_token=" +
req.query.access_token);
```

Normally this returns all events, but there is no point in fetching event information about an event that will start years ahead. Therefore two extra filter parameters are added to the query, "since" and "until". The response will then be a list of only ongoing events information in JSON format.

The Facebook event information always includes the address of the location which by itself would be relevant hashtags. The Hashtag API builds a large string with all the event descriptions and sends it to Alchemy Entity Extraction API. Alchemy will then categorize words and return them in JSON format. Last thing for the Hashtag API to do is to filter out categories that most likely could contain relevant hashtags.

After running a few test it was concluded that the categories Person, Organization, Holiday, and MusicGroup where good choices to start with. All the words and names will be filtered out and added to a JSON structured array. Any spaces will be removed and a "#" symbol will be added before each word or name are pushed to the array.

The array is then simply printed out as the HTTP response.

Creating the JSON structure:

```
for (index = 0; index < response.entities.length;
++index) {

var recivedType = response.entities[index].type;

if (recivedType == "Person" || recivedType ==
"Organization" || recivedType == "Holiday"||recivedType
== "MusicGroup") {

var tag = response.entities[index];

hashtags.entity.push({

"type" : tag.type,

"tag" : "#" + tag.text.replace(/\s+|\\n/g, '')

});

} }
```

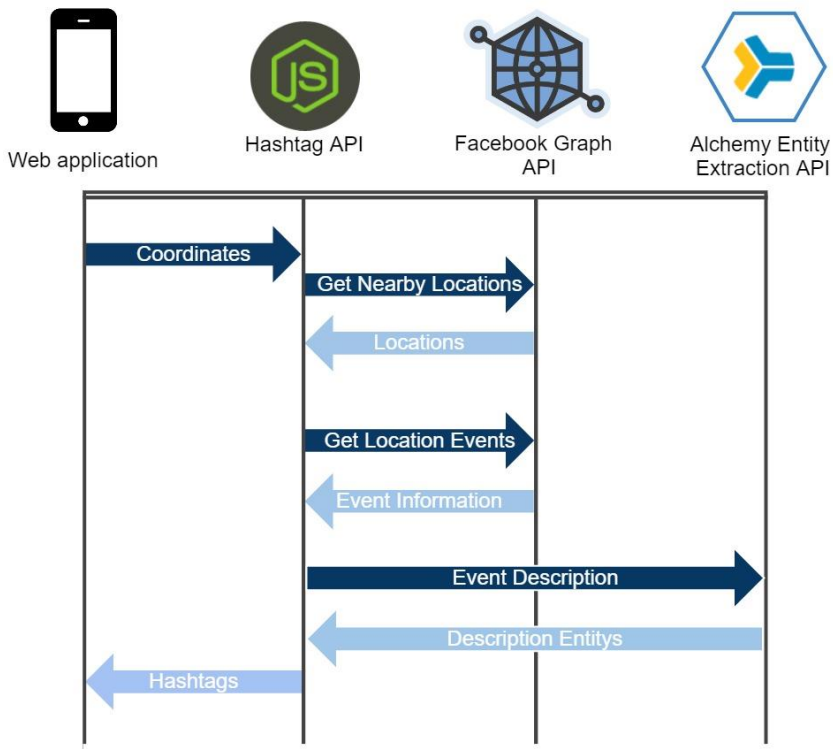

Fig. 7. Sequence diagram over the Hashtag API

5.4. Result of phase two

All the functionalities specified by the MVP was successfully developed in to the prototype interface.

Since the Hashtag API was developed as a Node.js application it could easily be uploaded and hosted on Bluemix. The API then receives a public web address so it can be accessed from anywhere. The web application can then make a HTTP request to that address with a query that specifies the coordinates and search distance.

For example using following query with coordinates to The O2 Arena in London with a search distance of 1km:

```
?lat=51.5030187&lng=0.0031647&distance=1000
```

Returns the following output:

```
{"entity":[{"type":"City","tag":"#London"},{"type":
"venue","tag":"#TheO2"}]}
```

The Hashtag API will always try and return two hashtags if it cannot find any ongoing events in the area. The two hashtags are then based on the nearest Facebook location instead. They are extracted from Facebook location city and location venue. In the example above no events where going on at the time of the test and therefore it could only return the hashtags #London and #TheO2.

Running another test with the coordinates for Tele 2 Arena in Stockholm during Eurovision Song Contest returns the following output:


```
{"entity":[{"type":"City","tag":"#Stockholm"},{"type":"venue","tag":"#Tele2Arena"},{"type":"Person","tag":"#SannaNielsen"},{"type":"Person","tag":"#DannySaucedo"},
{"type":"Company","tag":"#Tele2"},{"type":"Person","tag":"#Carola,Loreen"},{"type":"Person","tag":"#JohanBernhagen"},{"type":"Person","tag":"#PetraMede"},{"type":"Person","tag":"#TimHenri"}]}
```


The web application sent a request to the Hashtag API when the site was loaded. The resulting hashtags is then displayed in a separate field beneath the text field for the transcribed text. The user can then easily remove hashtags by clicking on them or add their own hashtags by typing in them manually. The hashtags are then added automatically to the end of the message when the user presses a button to post it.

6. Phase three - Backend API

The focus of the project changed unexcitingly when the MVP of the web application was ready. The task was now instead shifted into building backend APIs to support native Android and iOS applications. Therefore this third phase was added instead of finalizing the web application which is dedicated to the planning and development of the APIs.

6.1. Goal

The goal of this phase is to figure out which features from the MVP that are reasonable to be developed as separate APIs so that other devices also can have the possibility to access these features.

It was also necessary to make an assessment of the probability that the time schedule could be held and which features that should be given priority.

6.2. Planning and time assessment

Since the project had been going on for quite some time a new planning session with IBM was in order to assess the features from the final product or even the MVP could be done in time. And if this was not possible a decision on which features that should be prioritized and how to make the most of the time that was left. The issue was that although the group had some minor experience with creating APIs there was no time to get acquainted to the new service API connect that was asked by IBM to be used for this. During the meeting however engineers from IBM showed the basic functionality of the service which led to the conclusion that the project would be possible to produce the MVP during the timeframe that was left.

In order to reach the deadline the project was divided into three tasks:

6.2.1. Making the Hashtag API compatible with API connect

This is the same Hashtag API that was created for the web application, however some changes must be done in order to make it compatible with IBM's API Connect service.

6.2.2. Creating the Text Storage API

For the storage API the decision was between a conventional SQL database or a NoSQL database which consists of JSON objects instead. Since the Bluemix platform has native support for the Cloudant NoSQL Database a decision was made to use this for the storage API. To save data to the database a user sends information to the API as a JSON object in the format:

```
{
  "date": "2016-01-05",
  "userID": "DeviceID",
  "texts": "This is a test message",

}
```

To receive data the device sends a request to receive all saved data that belongs to a specific userID. The userID will be unique to every device so that a user would never receive other messages than the ones with the userID. This will be created inside IBM API connect.

6.2.3. Creating the Stenographer API with API connect

The Stenographer API will be combining the Watson speech to text API and the Alchemy entity extraction API, it will also have its own data storage to save the recorded sound. This is so that features for saving the recorded sound and to get hashtags based on the recorded audio would be possible through one API. The reason to create a separate API and add all these features was to make it easier for other developers. There is for example no longer need for a developer to specify initiation requests to Watson for every new program the developer makes. This is instead handled by the Stenographer API, which is a serious time saver.

It is also to be able to secure the API using the API connect service. This is because a direct web socket connection cannot be secured using API connect, a solution to make a separate API authenticated before establishing a connection to the Speech To Text service was deemed the best solution in the amount of time left on the project.

This will work by having a device send a stream of audio to the Stenographer API, then the Stenographer API will then record the audio whilst sending it along to the Watson speech to text service. The Stenographer API will then receive the text back from Watson speech to text and send it back to the device in the same JSON format. See figure 10 for a sequence diagram. When the device wants hashtags based on the text a request is sent to the Stenographer API, the text will then be analyzed with the Alchemy API entity extraction and sent back to the device as hashtags in a JSON format that is identical to the one used by the regular Alchemy API. Figure 8 can be used in order to clarify how the API is intended.

When it comes to the prioritization of these tasks the decision was that since the Hashtag API already done an only needed a slight alteration to function in the API connect service, this could be very good first priority. This could also be used to get accustomed to the API connect developing environment. Since the Watson speech to text feature can be used as an API already although not with the same features as wished for it was still put on the lowest priority which makes the priority in this order:

1. Integrating the Hashtag API with API connect
2. Creating the Storage API with API connect
3. Creating the Stenographer API with API connect

After the planning and prioritization was done a brief period was spent reading the documentation of the API connect to get acquainted with the developing environment.

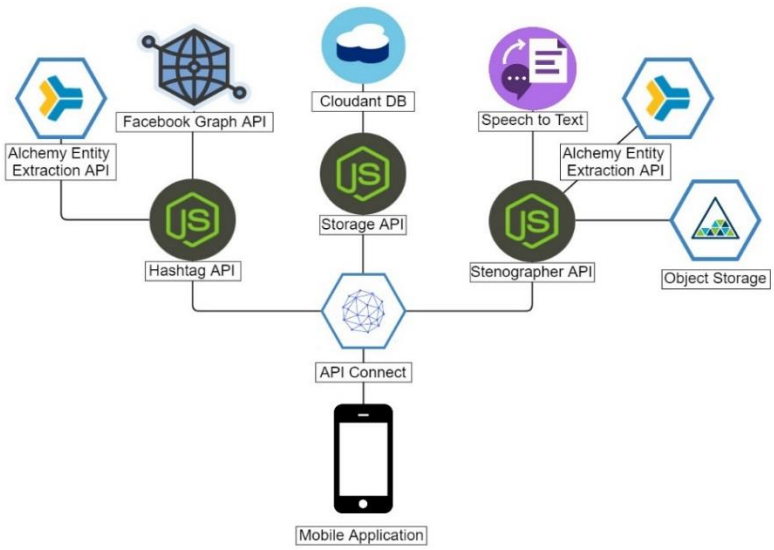

Fig. 8. Overview of the planned project

6.3. Integrating the Hashtag API with API Connect

The development began with the first priority and since the Hashtag API already was deployed in Bluemix the process was simply to change the flow of information so that information would go through API connect as a security measure instead of going directly to the Hashtag API. The reason this becomes a more secure alternative is because devices need to authenticate themselves before sending data to the API while using API connect. The authentication comes in the form of a traditional app secret and app id which is received by registering an application through the IBM developer cloud. If just a pure API without any security anyone could access it and basically use as much traffic as they wanted which could lead to the API crashing.

The hashtag API uses HTML queries to receive information and therefore the plan is to make sure that all HTML requests first go through the API connect service with an extra app id parameter for security.

To achieve this, a way for API connect to know where to send the data after authentication is needed, this could be done by creating a invoke inside the API connect designer that referenced to the hashtag API HTML request. Just sending the data to the invoker will not work however since the service does not know what to do with the values given, a way to instruct API connect on which place in the URL that the certain values should be placed has to be implemented. By defining the values inside the API connect designer as variables there is a possibility to use those variables inside the reference as such:

```
http://adresstohashtagapi.com/events$(request.search)
```

By doing this there is a possibility to secure the Hashtag API since it has to go through the authentication process from the API connectivity service, one can also use the statistics provided which includes how many simultaneous uses the API has and from which applications they request the API.

Inside API connect there is a function to automatically test the created APIs if they can use the get or post functions with the click of a button but in order to precisely troubleshoot the process the web application from the first phase was used as a test platform for the API' s instead.

Since the Hashtag API now was fully working and running with the API connect service, the development of the Storage API could begin.

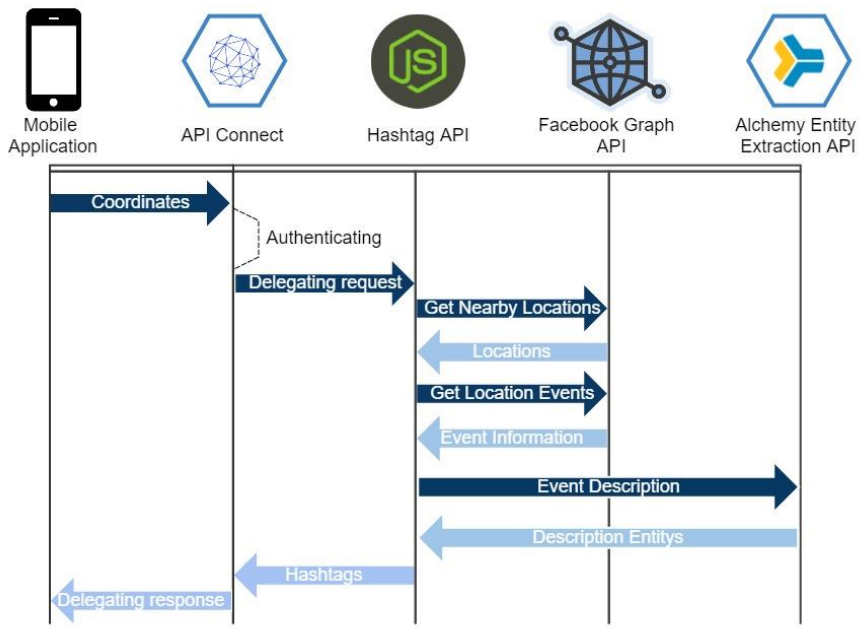

Fig. 9. Sequence diagram for the Hashtag API with authentication

6.4. Creating the storage API with API connect

As mentioned before the Storage API will use a NoSQL database for storage saved speeches in text form on the cloud, this is because the support for the Cloudant NoSQL database in both the Bluemix and API connect which should ease the development process.

The process to create the database and make it accessible through the cloud is a fairly easy thing to do since the database can be instantly created and started in the cloud. The difficulty lies in mapping the data in API connect so that data actually can be stored and received from the database in the cloud. This could be done in the API connect design view. When saving data it was only a matter of mapping the data of the values mentioned in 6.2.2 so that they all entered the same JSON object in the database. In order to retrieve an extra section for a search query was added so when a device tries to receive a text the database is searched for all texts with the device's unique userID and sends the all texts with the corresponding userID back to the user.

6.5. The process of creating the Stenographer API with API connect

The Stenographer API consists of multiple APIs working together, these are Watson speech to text (STT), Alchemy API entity extraction and the Object storage service from IBM. Work began with implementing the speech to text service piping since the other APIs would require this function to work before any implementation could be done.

Many might question the thought behind why a separate API that pipes the Speech to text data was implemented instead of using the original API, as mentioned in 6.2.3 one reason is because of the sound recording functionality. The goal is to have as few operations to run on the device itself and to run them on APIs instead so to make sure that the phone does not need to save the stream as a file and then later upload this to a database, a separate API to take the stream directly was needed.

Another critical reason is for the ease of the users or developers that will use the API, the original Watson speech to text API is very

customizable API with many different settings and tweaks that needs to be specified before being able to use it. While this is very nice for developers that need some special specification, it is very difficult to get running especially for people with little experience with how APIs work. The API created for this project will automatically set up the most used setting which is that the API will receive a sound stream and will give back the speech in text form, and after a 30 second period of inactivity the connection automatically disconnects. As a result of the API specifying these settings the only thing the developer using the Stenographer API has to worry about is to connect and to stream the audio which is a much easier process.

During the development a discovery was made that the web sockets needed to pipe the sound cannot be secured through the API connect service because web sockets does not allow to be blocked to wait for authentication but only opened or closed. That meant that a user could still stream audio and receive text from the service even though he or she has been rejected by the API.

To solve this a redesign was made to ensure some form of security, the new design was to make the Stenographer API without security outside of API connect and when this would function properly a token generator would be made inside API connect instead. The token received should be used as authentication for the Stenographer API instead. The engineers at IBM confirmed that this was a possible solution to the problem.

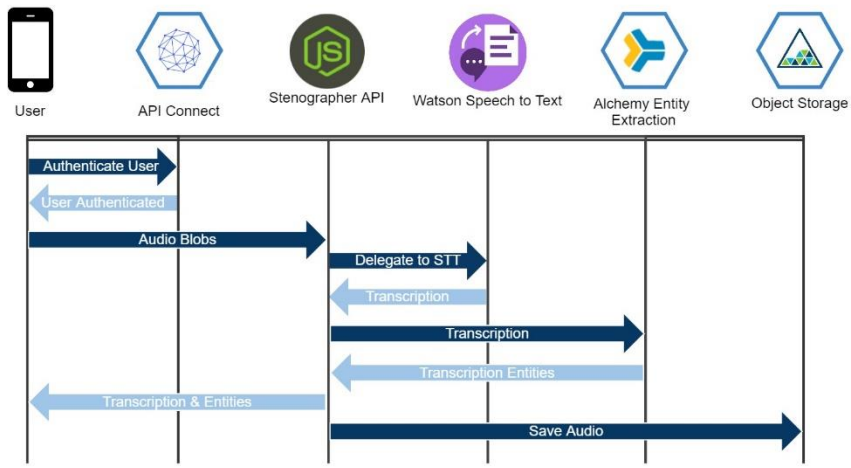

Fig. 10. Sequence diagram of the Stenographer API.

A decision was also made that the standalone API was to be developed in Node.js so that it would be developed in the same language as the Hashtag API and also because this was the language the group was most comfortable with.

During the development an important aspect was to keep track of syncing between the two pipes would be working and that no extra calculations would be done that would create a delay on the Speech to Text feature. In order to achieve this a clear indication to where data is flowing is needed. An example of this would be that the socket from the client side cannot send a sound stream at the same time as the Speech to Text API sends data back to the client socket, this needs to be done in different chunks that are sent in such a manner that it seems to happen at the same time for the user. This is thankfully fairly simple to do while using the "on receive data" and "on sending data" functions that are included with the web socket in Node.js since they handle this data exchange automatically. But if the developer is not careful these functions can fail to achieve to be asynchronous which would make the API meaningless if not capable of handling more that on user at the same time. This was a major difficulty and it was necessary to test every single iteration of the code was capable to handle multiple users and also that a user could close and open a socket without ant data

errors. The testing was done by starting multiple (usually 10) Speech to Text sessions and check that all of them got the correct result. Otherwise a specific error code would show up in the log which would specify in which general area the issue came from.

When the speech to text pipelining worked through the Stenographer API the Alchemy API entity extraction part was developed. This was pretty simple since the only thing needed was to send the recorded speech in text form to the alchemy API and the answer was sent right back, the difficulty here was also to maintain the asynchronous design to make it work in multiple uses simultaneously. The group decided that this could be decided by the developer of the app since hashtags is not always wanted when recording speech to text and therefore it would be a waste of processing power and space. So the API does not analyze the text and send back hashtags before the developer says so in the form of a signal flag, and when the API receives this flag it analyzes the text and sends the hashtags back. This does cause a delay of a couple of milliseconds for the speech to text feature but the API catches up quickly and is not noticeable by the naked eye.

The last function to implement was the Object storage which could save the recorded sound on a cloud database. Since the audio stream is in the standard WAV format the process of merging multiple sound blobs from the stream was fairly simple. The WAV format only contains some headers and binary data. In order to make a WAV file headers corresponding to those of a WAV file can be created beforehand and then the binary data from the stream can be added to make a large file, and when the stream has been ended by the user the complete sound file is uploaded to IBM's object storage service.

After all this the Complete API was stress tested to make sure that it could work with many simultaneous users which it could.

When all features where functioning fully however there was no time to implement the token part in API connect so the Stenographer API will be a standalone API.

In order to easier understand how this works a sequence diagram was made which can be seen in figure 10.

6.6. Result of Phase three

The final decision was that creating APIs of the different features was possible with the time left, and that it should be done with three separate APIs a storage API capable of storing and receiving saved speeches. A Speech to text API which also has the capabilities of giving hashtag suggestions based on the recorded text and also saves the recorded audio to a database, and finally the already created Hashtag API which tries to figure out at which event the user attends and who is speaking at the event and send that back as hashtags.

Technically any device capable of REST API handling should be able to access these but for this project an android and iOS device where the two test devices, and the APIs should be tested in the form of a separately created iOS application and android application.

All of the APIs that where planned for this phase where successfully developed with all of the mentioned features but with none of the possible extra features that could have been developed if there was enough time. The overall layout of the APIs looks a little bit different however as seen by comparing initial layout in figure 8 and the final layout in figure 11, this was because there was no knowledge on how the new API connect service worked initially but the functionality is still the same as expected.

The hashtag API and the storage API where able to function as planned with all the features that where specified and with no major difference in how the communication with the device will be handled.

There was a difference in the Stenographer API since web sockets cannot be protected using the API connect service, so a solution was made using a separate token generator that used the API connect service instead and the Stenographer API had to authenticate using that instead if a more secure API was wanted. The Token generator was skipped however due to lack of time to implement this feature and a standalone Stenographer API was made instead.

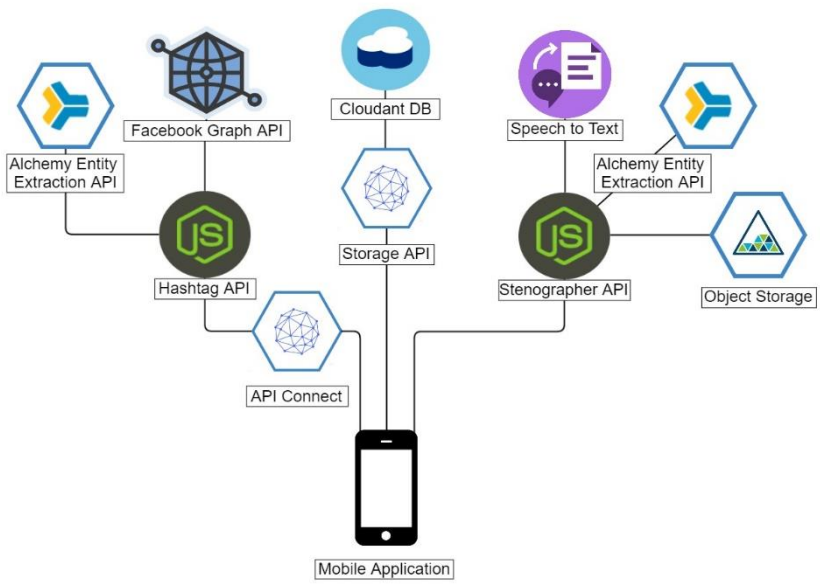

Fig. 11. Overview of the final result.

7. Evaluation

This section evaluates the different IBM services and also answer the questions stated in the problem specification (1.3).

7.1. IBM Speech to text

The IBM Watson Speech To Text service was very quick and responsive with audio streams which is a must for the project. There are also many features which are very useful in some cases such as the confidence variable that shows how confident the API is that the word is correct and which other word it could be instead. It can also detect if the person speaking is hesitating which can be very good for the user. There is also a large variety of services for developers such as automatic disconnection from the service info voice is not registered and the decision to use either a stream of sound or a file amongst many things. This high customization does make the service harder to use since the developer has to specify if the features should be used or not, while this might not sound strange it's very difficult when the documentation on how to do so is lacking. The fact that a decision on which language to use before recording can also cause some issues since people sometimes use words and phrases from other languages which the Speech to text service will treat as a word from the decided language which in some cases can give a completely different sentence after the entire word processing phase.

To identify a different language by just one word is a difficult task but would definitely improve the using experiences.

This mean that the service also tries to interpret inaudible sounds such as sneezes and cars passing by as words as well. This could be eliminated by using sound filtering software on the sound before processing the sound which would be a nice feature to have.

The Speech to text service is also very stingy when it comes to accents, when speaking English it seems to be fine with most American accents but struggles with other accents depending on how wide it is.

This is troublesome since that narrows down the amount of people that would find this useful.

The final verdict for the service is that it definitely has potential but in its current form the Speech to Text service from IBM is not polished enough to create a flawless experience for the end user.

7.2. Alchemy API (Entity extraction)

The alchemy API is a collection of different text analyzing services such as sentiment analysis and Concept analysis, this project only made use of the Entity extraction service and is therefore the only part that will be evaluated.

The entity extraction was well documented and easy enough so that usage for unexperienced developers goes without any issues, the results were also accurate. The API support multiple languages and recognizes the languages automatically within the text so that text containing multiple languages was possible.

The service was also fast enough to analyze the text and send the entities back so that no one testing the application ever complained on slow generating times.

The way to improve the service would be to make it more aware of the contexts in the text. An example is if the service analyzes the Swedish phrase "Här är dina biljetter"(Here's your tickets in English) The API will then state that a name was mentioned in the text and that name was "dina biljetter", it does this because Dina is a real name and the API directly assumes that the next word then would be the surname no matter the context. This is true for most words that also happens to be names.

The final verdict is that the Alchemy API definitely is capable to be used in these sorts of applications.

7.3. Cloudant NoSQL database

There are many advantages of using a NoSQL database instead of a conventional SQL database, the fact that multiple different JSON-objects can be stored in one database and that every put and get is sent through JSON objects are some of the advantages to name a few.

And using the Cloudant service on IBM Bluemix makes creating databases in the cloud very easy, but using a NoSQL database has flaws as well.

The first issue is since the NoSQL databases lacks a structure the developer has to specify how and what to search, to make this clearer we could use an example.

Let us say that we got two databases named database1 and databse2, database1 is a SQL database and database 2 is a NoSQL database and that both contain an id field and a name field.

To retrieve everything in the table that you have in database you simply need so make a query for the SQL database: Select * from table1. On the NoSQL database however you first have to specify to the database what the database should return if a person is searching and the also specify with what criteria this person will be searching.

This might not seem too hard but without proper documentation on how to do so it is very hard to actually get working.

There are also many ways of searching (called views) and the differences are never mentioned in any documentation found so the only way to know is to test it out.

The final verdict is that the Cloudant database definitely was capable for use in the project however the documentation was lacking.

7.4. API connect

The API connect service is intended to make the process of creating APIs and also to secure these APIs and the service definitely has potential. The graphical interface is easy to grasp and the way to add variables directly really help out. However since the service is very new there is a lot of bugs making it very time consuming. There is also not a lot of documentation to help if any trouble ever come up. The service has great potential if these issues are fixed.

In this project the API connect ended being a valuable tool despite its flaws. It greatly reduced the time taken to create secure APIs on Bluemix but it cannot be considered a tool that is simple to use.

7.5. Noodl

Noodl was used for prototyping the application before the project was in the development phase, the no coding interface can be very nice for designers that are unexperienced with coding and it also helps to "visualize" what the developer has done so far. There is also a possibility to add custom JavaScript's which enables the user to make more advanced prototypes both visually and functionally.

The interface is however a bit difficult to get accustomed to and sometimes required a lot more clicks than the competitors which should be fixed. The program was otherwise fully capable of doing what it was intended of doing and is therefore considered to definitely be capable of use in other projects.

8. Conclusion

8.1. How did the planning go?

Initially the group made a Gantt chart in order to evaluate how much time that could be spent on the project and how much the group would be able to achieve. However the sudden change of the project caused the Gantt chart to be scraped and a new plan was made after some discussions with the company.

The plan was to work in 1 week sprints and to only create the MVP that was defined in the beginning.

This new work plan did in fact hold without any compromises on the final product, however it was rather stressful and there was not enough time to implement any of the extra features that was discussed. One of the features was that the application should be able to recommend similar topics of interest to the user based on the speech. It was also discussed that the app should be able to tell the user if anyone commented on a post related to the subject.

8.2. Thoughts About the result

Even though the project was changed in the middle the group still managed to produce a product that passed the MVP which is very satisfying. But as developers it would always be nice to implement some extra features that at first we thought we could complete. It's a bit frustrating to not be able to further develop the product especially with so many ideas as the group has to improve it. The API was developed using the new API connect service which was a brand new service that had not yet been released yet and was very satisfying to be able to implement in the APIs without any kind of documentations from the company, it was also a very nice feature as developers to be able to view all the statistics given from API connect. There can also be a lot of value in securing the APIs with an app secret and app id the service provides.

The Speech To Text part works well in a stream and also works rather quick so that it would not be frustrating as a user, it is however frustrating that the service tries to translate all sounds to words because you sometimes get entirely different sentences.

When it comes to the storage part this works just as expected which is very nice, it is not always that programs actually works the way initially intended without any major modification. It also works very well in the final product and the app developers had no issues using this API in their application either. Both the android and the iOS device could use the APIs but only the android application was completed by the app team, it's sad that the iOS app never saw the light of day it was very reassuring to know that the APIs worked with both devices as we had planned from the beginning.

The Stenographer API was a fairly advanced program and was therefore very satisfying to see that the group managed to get it up and running. It was also good to see that it was so responsive even though we piped the speech and analyzed the text it as well.

8.3. Future development

8.3.1. Hashtag API

The Hashtag API is a tricky API since it has to be "smart" and deliver results that are relevant. But what is seen as relevant can change between users. There is no simple solution to this but one way could be to track what kind of tweets the user usually post and then prioritize that category. For example if the user often post names as hashtags the "Person" category could be prioritized and more hashtags of that kind would be shown.

The API currently only delivers hashtags that are generated from Facebook which has some drawbacks. One drawback is that the there is no standard format or content of the information presented in the event description. This means that it is not guaranteed that any hashtags can be extracted from the description or that they will be relevant. Ticnet API could be a nice complement to Facebook since their data is more structured.

8.3.2. Storage API

The Storage API is the most simple of the APIs and already has all the planned functionalities to support the apps.

One idea for future development would be to include a variable to store the coordinates for the recorded text. That would allow the app developers to also display the location of where the text was recorded. This could however already be done by saving the coordinates in the text field and removing it before displaying it to the user.

8.3.3. Stenographer API

The Stenographer API function to extract hashtags has room for improvement. One of the things that can be improved is to make the token API through API connect as it was intended in the first place.

There is also possibilities to improve the speed and stability of the API.

9. Terminology

Android	A operating system developed by Google and is focused on mobile devices.
AngularJS	Open-source client-side web application framework that aims to simplify both the development and testing of cross-platform mobile apps.
Apache Cordova	A mobile application development framework. Allows developers to code using CSS3, HTML5 and JavaScript instead of platform specific APIs.
API	Application Programing Interface. Routine definition, protocols and tools for building software and applications. A good API provides the building blocks and makes it easier for the programmer to develop.
Blackberry OS	Blackberry's own operating system for their smartphone devices.
CRUD	Create, Read, Update and Delete. An API with CRUD capabilities can be used to directly interact with a database to for example update existing data objects.
CSS	Cascading Style Sheets. A language that describes the presentation of a document, often used in combination with HTML.
FQL	Facebook Query Language, a SQL-style interface to query the data exposed by the Graph API.
Group	In the thesis there are many references to the group, this means the two authors of this thesis.

HTML	HyperText Markup Language is a markup language for creating web pages.
HTTP	Hyper Transfer Protocol. A Communication protocol that is used to transfer webpages on the internet.
iOS	Apples mobile operating system for Iphone, Ipad, Ipod touch and Apple TV.
JSON	JavaScript Object Notation. Open-standard format that uses readable text to transfer data.
LUBsearch	A collective search engine for all libraries that the University of Lund is affiliated with.
MongoDB	A NoSQL open-source cross-platform document-oriented database.
Oauth 2.0	Authorization framework enables a third-party application to obtain limited access to an HTTP service.
Platform as a service (PaaS)	A category of cloud computing services that provides a platform allowing customers to develop, run and manage applications.
Responsive web design	When a webpage automatically formats its interface to fit different devices.
REST	Representational State Transfer is an architectural style that is communication oriented and often used in the development of Web services.
SDK	Software Development Kit. Set of software development tools that allows the creation of applications for a certain software package.
SOAP	Storage Object Access Protocol A protocol specification for exchanging structured

	information in the implementation of web services in computer network.
SQL	Structured Query Language is a program language for receiving and modifying data in a relational database.
TCP	Transmission Control Protocol is a communication oriented data transfer protocol that is used in most of the communication over internet.
URL	Uniform Resource Locater. Usually called web address which identifies a specific resource on the internet, for example a web page.
VCAP	VCAP is an Environment variable in JSON format with information that is used to interact with a service instance in Bluemix.
WebSocket	A protocol that allows full-duplex communication over a single TCP connection.

References

[1] "IBM Brings Watson Apis to Bluemix Paas," [Online]. Available: http://www.datacenterknowledge.com/archives/2014/10/08/watson-comes-to-ibm-bluemix-paas/. [Accessed 14 April 2016].

[2] "Strongloop," [Online]. Available: https://strongloop.com/strongblog/introducing-ibm-api-connect/. [Accessed 17 April 2016].

[3] "Loopback," [Online]. Available: https://loopback.io/. [Accessed 17 April 2016].

[4] IBM, "StrongLoop and Bluemix," [Online]. Available: https://www.ibm.com/developerworks/community/blogs/dfa2dc54-5a14-4cf8-91e0-978bfd59d0d4/entry/What_is_StrongLoop_and_how_does_it_relate_to_Bluemix?lang=en. [Accessed 17 April 2016].

[5] "IBM API Management," [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSWHYP_4.0.0/com.ibm.apimgmt.overview.doc/api_management_overview.html. [Accessed 02 04 2016].

[6] "AlchemyAPI," [Online]. Available: http://www.alchemyapi.com/. [Accessed 18 April 2016].

[7] "AlchemyAPI," [Online]. Available: http://www.alchemyapi.com/api/entity/types. [Accessed 18 April 2016].

[8] "Facebook Graph API," [Online]. Available: https://developers.facebook.com/docs/graph-api. [Accessed 20 April 2016].

[9] Caleffi.P, The 'hashtag': A new word or a new rule?, 2015.

[10] T. &. Vinosku.S, Node.js: Using JavaScript to Build High-Performance Network Programs, 2010.

[11] "Ionic framework," [Online]. Available: http://ionicframework.com/. [Accessed 25 April 2016].

[12] "Express framework," [Online]. Available: http://expressjs.com/. [Accessed 25 April 2016].

[13] Skolverket, "Källkritik på internet," [Online]. Available: http://www.skolverket.se/skolutveckling/resurser-for-larande/kollakallan/kallkritik/fakta/internet-1.192625. [Accessed 22 April 2016].

[14] C. A., The inmates are running the asylum: why high tech products drive us crazy and how to restore the sanity., 2014.

[15] R. M. D., Minimum Viable Product and the Importance of Experimentation in Technology Startups, 2012.

[16] "Web SQL Databases," [Online]. Available: http://html5doctor.com/introducing-web-sql-databases/. [Accessed 15 May 2016].

[17] L. R. F. J. a. R. L. M. T. o. P. A. a. M. I. V. 5. (. 1. p. 1.-1. Bahl, A Maximum Likelihood Approach to Continuous Speech Recognition., 1983.

10. Appendices

10.1. Speech to text publications

1. Bahl, Lalit R., Frederick Jelinek, and Robert L. Mercer. *A Maximum Likelihood Approach to Continuous Speech Recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 5(2) (March 1983): pp. 179-190.
2. Hinton, Geoffrey, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups.* Signal Processing Magazine, IEEE, Vol. 29(6) (November 2012): pp. 82-97.
3. Jelinek, Frederick. *The Development of an Experimental Discrete Dictation Recognizer.* Proceedings of the IEEE, Vol. 73(11) (November 1985): pp. 1616-1624.
4. Padmanabhan, Mukund, and Michael Picheny. *Large-Vocabulary Speech Recognition Algorithms.* Computer, Vol. 35(4) (2002): pp. 42-50.
5. Picheny, Michael, David Nahamoo, Vaibhava Goel, Brian Kingsbury, Bhuvana Ramabhadran, Steven J. Rennie, and George Saon. *Trends and Advances in Speech Recognition.* IBM Journal of Research and Development, Vol. 55(5) (October 2011): pp. 2:1-2:18.
6. Soltau, Hagen, George Saon, and Tara N. Sainath. *Joint Training of Convolutional and Non-Convolutional Neural Networks.* Proceedings of the 2014 IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), Florence, Italy (May 2014): pp. 5572-5576.

10.2. How hashtags are used on social media

Hashtags are used very often as a contextual aside to comment on, give more depth to, or somehow emphasize what has been said, as in:

- Sarah Palin for President?? #Iwouldratherhaveamoose
- My bestie has the best Instagram. Would it be weird if I started having her edit all my photos? #kidding #butnotreallykidding
- My arms are getting darker by the minute. #toomuchfaketan

but also as a disclaimer:

- BREAKING: US GDP growth is back! #kidding

as a (seemingly) accidental remark or naming:

- Ahahahah Jack comunque ti tradisce... con mio fratello #ops

to express personal feelings and emotions:

- #angry

to support events or movements:

- #PrayforBoston

for self-mockery:

- Feeling great about myself till I met an old friend who now races at the Master's level. Yup, there's today's #lessoninhumility

for brand promotion:

- #ShareaCoke

for chat/conference participation:

- #ESSEconference

[9]

10.3. Facebook Graph API response

10.3.1. Event description Science Museum London:

{
"description": "This event goes on sale at 10.00 on Friday 12 February 2016

Join us for a special talk at the Science Museum's IMAX Theatre between astronaut Buzz Aldrin and British physicist Brian Cox.

Aldrin was selected by NASA in 1963 into the third group of astronauts and on 20 July 1969 made history with Neil Armstrong during their Apollo 11 moonwalk, becoming the first two humans to set foot on another world.

Since retiring from NASA, Aldrin continues to chart a course for future space travel and is passionate about inspiring the younger generations of future explorers and innovators.

Dr. Aldrin is an author of nine books including 'Mission to Mars: My Vision for Space Exploration' which outlines his plan to get us beyond the moon and on to Mars. He continues to inspire today's youth with his illustrated children's books including 'Welcome to Mars: Making a Home on the Red Planet'.

Professor Brian Cox is an Advanced Fellow of particle physics in the School of Physics and Astronomy at the University of Manchester and presenter of numerous science programmes including BBC2's Stargazing Live.

The event will include an opportunity for the audience to ask questions and Aldrin will also be signing copies of his latest books. These will be available to buy from the Science Museum.

The talk is the latest in a series of events around Cosmonauts which celebrates some of the most significant moments of space travel history. www.sciencemuseum.org.uk/cosmonauts

Tickets
£10",
  "end_time": "2016-02-28T15:00:00+0000",
  "name": "Buzz Aldrin in Conversation with Brian Cox",
  "place": {
    "name": "Science Museum",
    "location": {
      "city": "London",
      "country": "United Kingdom",
      "latitude": 51.497292992767,
      "longitude": -0.17472351631983,
      "street": "Exhibition Road",
      "zip": "SW7 2DD"
    },
    "id": "7408594675"
  },
  "start_time": "2016-02-28T14:00:00+0000",
  "id": "419183491601291"
}

10.3.2. O2 Arena London:

{
  "events": {
    "data": [
      {
        "description": "** Note that this event has been rescheduled to 31 May 2017 **

The Sessions - A live re-staging of The Beatles at Abbey Road Studios - part blockbuster stage show, part access-all-areas musical documentary - will embark on an eleven-date UK arena tour in spring 2016 ending with a show at The O2 on Wednesday 31 May 2017.

Set in a state-of-the-art reproduction of the iconic Abbey Road Studio 2, The Sessions will stage breath-taking, musically spectacular new live renditions of the timeless albums recorded there by The Beatles, to take us on a joyous, thrilling, historically authentic voyage through the events that shaped popular music history.",


```
      "name": "The Sessions at The O2 arena",
      "place": {
        "name": "The O2",
        "location": {
          "city": "London",
          "country": "United Kingdom",
          "latitude": 51.502588335251,
          "longitude": 0.0043592565242575,
          "street": "Peninsula Square",
          "zip": "SE10 0DX"
        },
        "id": "57843015021"
      },
      "start_time": "2017-05-31T18:30:00+0100",
      "id": "102521466794156"
    },
    {
```


"description": "After six sell out UK tours, 300,000 tickets sold, countless television performances and numerous other accomplishments; Diversity have announced their seventh tour “Genesis” for April 2017.

After wowing audiences in 2015 with their 31 date Up Close and Personal tour, Diversity will once again return to The O2 arena. The new tour Genesis is the concluding part of their super hero fantasy epic and follows ‘Limitless: The Reboot’. Creator and choreographer Ashley Banjo said of the tour;

After finishing his fifth series as head judge on Sky 1’s Got to Dance, in 2016 Ashley Banjo currently co presents BBC One’s brand new entertainment programme ‘Can’t Touch This’ and will also front a pioneering television programme for ITV tackling the sensitive issue of bullying. Other Diversity members Jordan Banjo and Perri Kiely continue to work closely with Nickelodeon and have just returned from Los Angeles as UK presenters for the Kids Choice Awards for the third consecutive year. Diversity continue to inspire the next generation of dancers as they will be teaching over 150,000 children in their academies at Butlins throughout 2016. Diversity will also be hitting TV screens throughout the year with jaw dropping performances not to be missed.",


```
    "name": "Diversity Genesis 2017 at The O2 arena",
    "place": {
      "name": "The O2",
      "location": {
        "city": "London",
        "country": "United Kingdom",
        "latitude": 51.502588335251,
        "longitude": 0.0043592565242575,
        "street": "Peninsula Square",
        "zip": "SE10 0DX"
      },
      "id": "57843015021"},
    "start_time": "2017-04-14T17:00:00+0100",
    "id": "1251783544835223"
  }
```

10.4. Alchemy Entity Extraction response

10.4.1. Science Museum London

A response to the event description in appendices 11.3.1

{
"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": "",
"totalTransactions": "2",
"language": "english",
"text": "This event goes on sale at 10.00 on Friday 12 February 2016 \n \nJoin us for a special talk at the Science Museum's IMAX Theatre between astronaut Buzz Aldrin and British physicist Brian Cox. \n \nAldrin was selected by NASA in 1963 into the third group of astronauts and on 20 July 1969 made history with Neil Armstrong during their Apollo 11 moonwalk, becoming the first two humans to set foot on another world. \n \nSince retiring from NASA, Aldrin continues to chart a course for future space travel and is passionate about inspiring the younger generations of future explorers and innovators. \n \nDr. Aldrin is an author of nine books including 'Mission to Mars: My Vision for Space Exploration' which outlines his plan to get us beyond the moon and on to Mars. He continues to inspire today's youth with his illustrated children's books including 'Welcome to Mars: Making a Home on the Red Planet'. \n \nProfessor Brian Cox is an Advanced Fellow of particle physics in the School of Physics and Astronomy at the University of Manchester and presenter of numerous science programmes including BBC2's Stargazing Live. \n \nThe event will include an opportunity for the audience to ask questions and Aldrin will also be signing copies of his latest books. These will be available to buy from the Science Museum. \n \nThe talk is the latest in a series of events around

Cosmonauts which celebrates some of the most significant moments of space travel history. www.sciencemuseum.org.uk/cosmonauts \n \nTickets \n£10\", \n",
  "entities": [
    {
      "type": "Person",
      "relevance": "0.88434",
      "sentiment": {
        "type": "positive",
        "score": "0.628848"
      },
      "count": "6",
      "text": "Buzz Aldrin",
      "disambiguated": {
        "subType": [
          "Astronaut",
          "AwardWinner",
          "HallOfFameInductee",
          "MilitaryPerson",
          "OperaCharacter",
          "FilmActor",
          "TVActor"
        ],
        "name": "Buzz Aldrin",
        "website": "http://www.buzzaldrin.com/",
        "dbpedia": "http://dbpedia.org/resource/Buzz_Aldrin",
        "freebase": "http://rdf.freebase.com/ns/m.0hfml",
        "opencyc":
"http://sw.opencyc.org/concept/Mx4rwRbTgJwpEbGdrcN5Y29ycA",
        "yago": "http://yago-knowledge.org/resource/Buzz_Aldrin"
      }
    },
    {

```
      "type": "Person",
      "relevance": "0.322441",
      "sentiment": {
        "type": "positive",
        "score": "0.320591"
      },
      "count": "3",
      "text": "Brian Cox",
      "disambiguated": {
        "subType": [
          "Academic",
          "Scientist",
          "TVActor"
        ],
        "name": "Brian Cox (physicist)",
        "website": "http://www.apolloschildren.com/",
        "dbpedia": "http://dbpedia.org/resource/Brian_Cox_(physicist)",
        "freebase": "http://rdf.freebase.com/ns/m.0g19ct",
        "yago": "http://yago-
knowledge.org/resource/Brian_Cox_(physicist)"
      }
    },
    {
      "type": "Facility",
      "relevance": "0.29595",
      "sentiment": {
        "type": "positive",
        "score": "0.512029"
      },
      "count": "2",
      "text": "Science Museum",
      "disambiguated": {
        "subType": [
```

```
          "Organization",
          "Location",
          "Building",
          "Museum"
        ],
        "name": "Science Museum of Minnesota",
        "website": "http://www.smm.org",
        "dbpedia":
"http://dbpedia.org/resource/Science_Museum_of_Minnesota",
        "freebase": "http://rdf.freebase.com/ns/m.04_bvy",
        "yago": "http://yago-
knowledge.org/resource/Science_Museum_of_Minnesota"
      }
    },
    {
      "type": "Organization",
      "relevance": "0.25614",
      "sentiment": {
        "type": "neutral"
      },
      "count": "2",
      "text": "NASA",
      "disambiguated": {
        "subType": [
          "Company",
          "GovernmentAgency",
          "AirportOperator",
          "AwardPresentingOrganization",
          "SoftwareDeveloper",
          "SpaceAgency",
          "SpacecraftManufacturer"
        ],
        "name": "NASA",
```

```
      "geo": "38.88305555555556 -77.01638888888888",
      "website": "http://www.nasa.gov/home/index.html",
      "dbpedia": "http://dbpedia.org/resource/NASA",
      "freebase": "http://rdf.freebase.com/ns/m.05f4p",
      "opencyc":
"http://sw.opencyc.org/concept/Mx4rwQwtspwpEbGdrcN5Y29ycA",
      "yago": "http://yago-knowledge.org/resource/NASA"
     }
    },
    {
     "type": "FieldTerminology",
     "relevance": "0.207822",
     "sentiment": {
      "type": "positive",
      "score": "0.293824"
     },
     "count": "1",
     "text": "Space Exploration"
    },
    {
     "type": "Person",
     "relevance": "0.200766",
     "sentiment": {
      "type": "neutral"
     },
     "count": "1",
     "text": "Neil Armstrong",
     "disambiguated": {
      "subType": [
       "Astronaut",
       "AwardWinner",
       "MilitaryPerson"
      ],
```

```
      "name": "Neil Armstrong",
      "website": "http://www.jsc.nasa.gov/Bios/htmlbios/armstrong-
na.html",
      "dbpedia": "http://dbpedia.org/resource/Neil_Armstrong",
      "freebase": "http://rdf.freebase.com/ns/m.05b6w",
      "opencyc":
"http://sw.opencyc.org/concept/Mx4rvrxFOZwpEbGdrcN5Y29ycA",
      "yago": "http://yago-knowledge.org/resource/Neil_Armstrong"
     }
    },
    {
     "type": "JobTitle",
     "relevance": "0.182202",
     "sentiment": {
      "type": "positive",
      "score": "0.512029"
     },
     "count": "1",
     "text": "physicist"
    },
    {
     "type": "Facility",
     "relevance": "0.179811",
     "sentiment": {
      "type": "positive",
      "score": "0.512029"
     },
     "count": "1",
     "text": "IMAX Theatre"
    },
    {
     "type": "Organization",
     "relevance": "0.169509",
```

```
  "sentiment": {
    "type": "positive",
    "score": "0.224871"
  },
  "count": "1",
  "text": "School of Physics and Astronomy"
},
{
  "type": "JobTitle",
  "relevance": "0.150669",
  "sentiment": {
    "type": "positive",
    "score": "0.224871"
  },
  "count": "1",
  "text": "Professor"
},
{
  "type": "Organization",
  "relevance": "0.147568",
  "sentiment": {
    "type": "positive",
    "score": "0.224871"
  },
  "count": "1",
  "text": "University of Manchester",
  "disambiguated": {
    "subType": [
      "Location",
      "CollegeUniversity",
      "ComputerDesigner",
      "University"
    ],
```

```
        "name": "University of Manchester",
        "geo": "53.465555555555554 -2.233611111111111",
        "website": "http://www.manchester.ac.uk/",
        "dbpedia":
"http://dbpedia.org/resource/University_of_Manchester",
        "freebase": "http://rdf.freebase.com/ns/m.0lbfv",
        "opencyc":
"http://sw.opencyc.org/concept/Mx4rwKT4sghoQdeN-
qdVKSpLDg",
        "yago": "http://yago-
knowledge.org/resource/University_of_Manchester"
      }
    },
    {
      "type": "Quantity",
      "relevance": "0.147568",
      "sentiment": {
        "type": "neutral"
      },
      "count": "1",
      "text": "10"
    }
  ]
}
```


10.5. Speech to Text example response


```
    }{
  "results": [
    {
      "alternatives": [
        {
          "transcript": "several tornadoes to "
        }
      ]
```

```
        "final": false
      }
    ],
    "result_index": 0
}{
    "results": [
      {
        "alternatives": [
          {
            "transcript": "several tornadoes touch "
          }
        ]
        "final": false
      }
    ],
    "result_index": 0
}{
. . .
}{
    "results": [
      {
        "alternatives": [
          {
            "confidence": 0.8691263198852539,
            "transcript": "several tornadoes touch down is a line of
severe thunderstorms swept
 through colorado on sunday "
          }
        ]
        "final": true
      }
    ],
    "result_index": 0
}
```