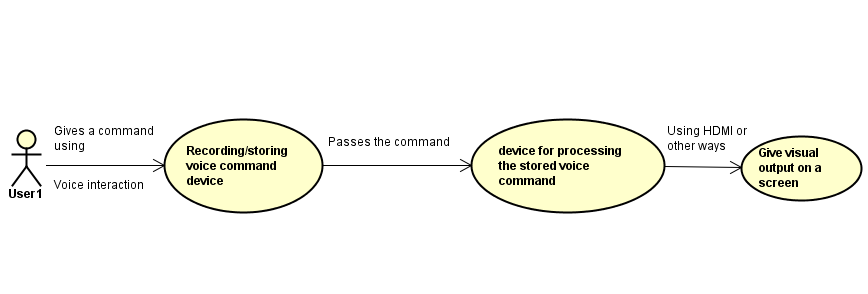
# Introduction

## Background Description

Oftentimes in their daily lives people find themselves wanting to have a device that can be controlled without a physical interface. One such situation would be when people are in their cars, driving, and they do not want to divert attention from the road. Another such situation is at home, when people often have their hands full when dealing with certain situations and do not want to interrupt what they are doing, go and search something on the internet. In such situations, a computational device with which one can interact hands-free is needed. Thus, the project aims to create a system which takes audio input from users and interprets it into commands that can be executed and whose outcome can be observed on a screen.

Here is a rich picture that describes the idea of the project and the direction of the it’s workflow.



## Scenarios

### House scenario – news

Imagine yourself at home, cooking in the kitchen. Everything is going along nicely when you glance at the clock on the wall and you realize it is 4 o’clock and the news are on. The first thing you think of is where is the remote, but then you realize that your hands are dirty and that the food is not done yet. If you go to the bathroom, wash your hands, then turn on the TV, you’ll have to get your hands dirty again. Not to mention that food does not wait for the reporter to finish the story, which means that when the chicken comes out from the over, or when the soup needs to be stirred, or when the table need to be set you are not going to hear the story.

### House scenario – music

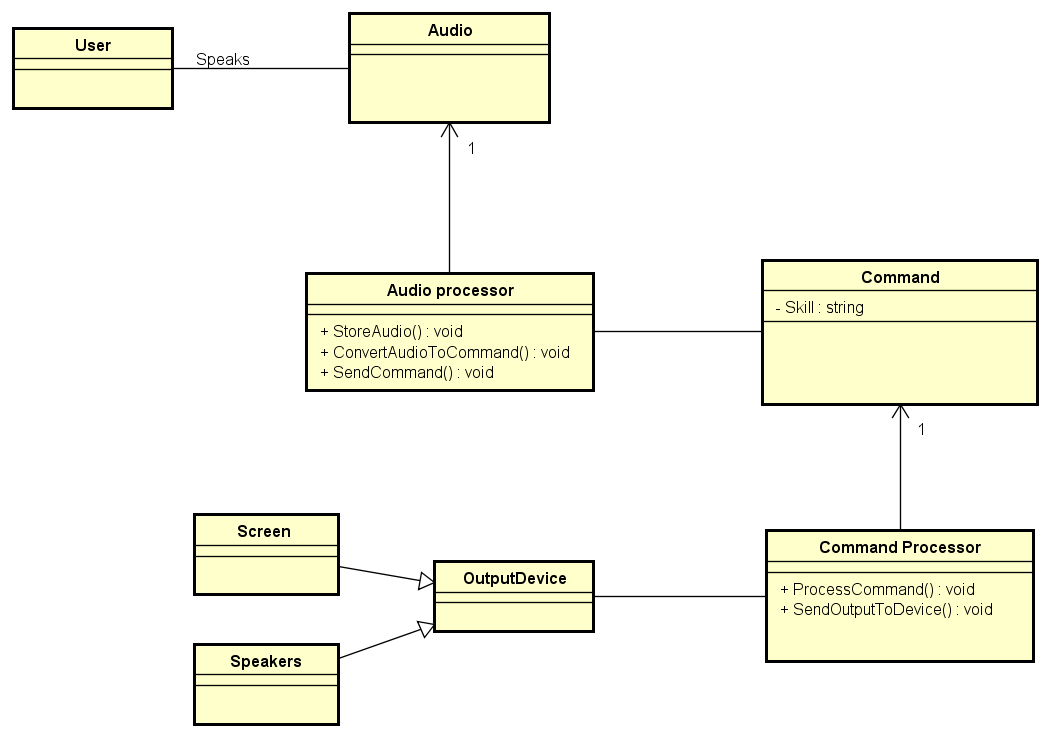
Imagine yourself again at home. You are cooking in the kitchen, but this time it not 4 o’clock yet, so the meal preparations are in full swing. Vegetables are flying left and right, a pot of water is happily sitting on the stove, both the table and the counter are covered in plates, cutlery and kitchen towels, but everything is in its place. Suddenly you remember that earlier, when you were driving home, they put your favorite song on the radio but you didn’t get to listen to it to the end, so you want to listen now. The laptop is in the living room, surely you could just search for the song on YouTube, but now your hands are full, literally, and there’s no sign you would have enough time anytime soon.

### Car scenario – music

This time imagine yourself driving. You’re coming home from work. It’s a 30 minutes’ commute, and while the road conditions are not particularly challenging, a certain amount of attention is needed in order to drive safely. Right about when you encounter a longer stretch of road you find yourself wanting to listen to some relaxing music. You even have a song in mind. The problem is that while the car has a touchscreen and a state-of-the-art entertainment system, you can’t really afford to take your eyes off the road long enough to use any of them. You don’t want to stop just for that either, as you’re not far from home and you’d rather keep going than go through the process of finding a place to stop, finding the song, and then reentering traffic.

# 2. Analysis

Based on the Scenarios earlier described the team made an idea about how the project should be structured. The project is broadly described with the help of an Abstract Class Diagram pictured below:



From this Abstract Class diagram the team could figure out what technical details, data manipulation, processing and calculations are needed and then how to structure the core system. More of this can be found in the next subchapter which will cover both functional and non-functional requirements as system delimitations.

## Requirements

These are the requirements which have been chosen after deliberating with the team, *STIBO Supervisor* and *VIA Supervisor*. The functional requirements are meant to specify one or more particular results of the system. They are then supported by the non-functional requirements which impose constrains on the implementation and design of the system such as performance, security, or reliability. Both functional and non-functional requirements are explained in detail in their particular sections.

### Functional requirements

A requirement that is crucial to the success of the project is the capability of the system to translate voice commands into text and then text into actual commands which need to be run. The finished system must be able to accept voice commands, process them and turn them into text, process the resulting text and turn it into commands, run the resulting commands and find the appropriate content, then displaying said content on a screen. Should the voice commands be interpreted into commands pertaining to a video output that is already being displayed (such as playback commands: “Pause”, “Play”, “Replay”, etc.), the respective commands’ results must be visible on the video content on the screen (the displayed video must be paused, playback must be resumed, the video must be replayed, etc.).

1. The system must convert human voice input into text.
2. The system must convert the text into a command.
3. The system must execute the command and display output.
4. The system must only interact with the user when the user specifically interacts with the system.
5. The system must have a Graphical User Interface.
6. The system must allow the users to search YouTube videos using voice commands.
7. The system must allow the users to have voice over media control over videos.
8. The system must allow the users to have voice over Web Page control.
9. The system must allow users to have multiple profiles.
10. The system must allow users to switch between profiles using voice commands.
11. The system must allow the users to change the Graphical User Interface.

### Non-functional requirements

1. The system shall ensure an output regardless of the environment in which it is used.
2. The system shall correctly convert voice input into text in at most 0.5s.
3. The system shall correctly convert text into a command in at most 0.5s.
4. The system shall correctly convert voice input into text in at least 80% of the cases.
5. The system shall correctly convert text into a command in at least 80% of the cases.
6. The system shall filter video output in relation to which user is using the system.
7. The system shall not check whether a user logs in their account and not someone else’s.

### Delimitations

1. The system will accept and interpret commands only in English.
2. The system’s output will only be visible from a single web-page application.

## Use Cases

To clarify and organize the system requirements the team made up a set of possible sequences of interactions between the system and users in a particular environment and related to a particular goal.   
This was done using Use Case Diagrams and Descriptions.  
The following image represents the full functionality of the system.

!!!Image here with a full functionality Use Case Diagram!!!

The Use Case Descriptions represent the following content. Only two of the most important use cases will be presented here. The first Use Case will concern the speech conversion to text and then to a command, as the second focuses on the usage of this speech command to see a YouTube Video. The other Use Cases as well as their descriptions can be found in Appendix C – Diagrams and Descriptions.

1.2.1. Display words on Web Page (kind of a simple name but we can change it)

This Use case emerged by grouping the four of the functional requirements which are:

* The system must convert human voice input into text.
* The system must convert the text into a command.
* The system must execute the command and display output.
* The system must have a Graphical User Interface.

The only purpose of the GUI mentioned here is to have an interface where the user can see the spoken words. It will have greater significance as the system gains more functionality from the other Use Cases.

The image below represents one part of the full system Use Case diagram.

!!!One picture here with this part only!!!

**Use case description**

|  |  |
| --- | --- |
| **Use Case:** | **Display Words on Web Page** |
| **Summary:** | This use-case describes a user speaking a phrase to the system, which will then be displayed as a text version of that phrase on Web Page which can be seen on a screen. |
| **Actor:** | **User** |
| **Precondition(s):** | The user must have a stable internet connection to the server.  The user must have the system started. |
| **Post condition(s):** | The system successfully converts the users spoken phrase into text.  The system successfully converts the text into the Display Command.  The system successfully displays the user’s phrase after the keyword on the Web Page. |
| **Base sequence:** | 1. User pronounces display function activation command followed by the words he would like to see on the Web Page. 2. The result will be displayed on the screen with the help of the GUI. |
| **Branch Sequence:** |  |
| **Exception Sequence:** | **The command can fail to be processed by the server due to internet connection errors.**   1. The Web Page browser will display the appropriate error |
| **Sub Use Case:** | **None** |
| **Actor Description:** | A person who is using the system |

1.2.2. Search YouTube Videos by keywords

This Use Case emerged from the same four requirements used for the previous “Display Words on Web Page” but the result will be different from the previous one in which a user would only see the words displayed. The GUI has been improved to give the user a good feeling when browsing videos on YouTube and the new command phrases which will provide the necessary functionality have been made.

One picture here with this particular part taken from the full system Use Case Diagram.

**Use case description**

|  |  |
| --- | --- |
| **Use Case:** | **Search YouTube Videos by keywords** |
| **Summary:** | This use-case describes a user asking the system to search for a video on YouTube by specific keywords. |
| **Actor:** | **User** |
| **Precondition(s):** | The user must have a stable internet connection to the server  The user must have the system started. |
| **Post condition(s):** | The system successfully converts the users spoken phrase into text.  The system successfully converts the text into the YouTube Videos Command.  The system successfully displays the user’s YouTube Video and the keywords used for searching this Video. |
| **Base sequence:** | 1. User pronounces Search on YouTube function activation command followed by the keywords meant for searching on YouTube. 2. The resulting video will be displayed on the screen with the help of the GUI. 3. The resulting video will play automatically. 4. The search bar will display the keywords used for this search. |
| **Branch Sequence:** | * Video does not exist:   The video frame will display an appropriate error message.   * Video cannot be reached:   The video frame will display and appropriate error message. |
| **Exception Sequence:** | **The command can fail to be processed by the server due to internet connection errors.**  1. The Web Page browser will display the appropriate error |
| **Sub Use Case:** | **None** |
| **Actor Description:** | A person who is using the system |

1.2.3. Media Controls

This Sub Use Case emerged after completing the “**Search YouTube Videos by keywords”** Use Case. Simply just giving the user the possibilty to search and autoplay a YouTube video contrained him from manipulating the output. Thus, this Sub Use Case was made in order to give the user the possibilty of choosing when to Stop/Play/Pause, etc by using new voice over commands.

One picture here showing the previous Use Case picture and adding to it the Sub Use Case.

**Sub Use case description**

|  |  |
| --- | --- |
| **Sub Use Case:** | **Media Controls For Videos** |
| **Summary:** | This use-case describes a user using voice commands to use media controls on a video such as Play/Stop/Pause |
| **Actor:** | **User** |
| **Precondition(s):** | The user must have a stable internet connection to the server  The user must have the system started. |
| **Post condition(s):** | The system successfully converts the users spoken phrase into text.  The system successfully converts the text into the commands used for Media Control. |
| **Base sequence:** | 1. |
| **Branch Sequence:** |  |
| **Exception Sequence:** | **The command can fail to be processed by the server due to internet connection errors.**  1. The Web Page browser will display the appropriate error |
| **Sub Use Case:** | **None** |
| **Actor Description:** | A person who is using the system |

!!! I think that if we manage to make a demonstration of these 2 Use Cases and the Sub Use Case we should be just fine!!! Of course, all Use Cases could be added here but I consider that these should be enough, considering that we have to take them through the implementation and testing phase.

We need one Activity Diagram to show the workflow of the “**Search YouTube Videos by keywords”** Use Case. We also need to describe it a bit.

1.3. Activity Diagram

With the purpose of giving a more detailed diagram about how a speicific Use Case acts from the starting point until it finished the process the team decided to show an Activity Diagram which will reflect the workflow of the second Use Case described in this report which is “**Search YouTube Videos by keywords”**.

!!!Picture here with Activity Diagram!!!

# 3. Design

As one of the most important tasks of this project was to use a speech recognition techonologie, the team had to make choice considering all the possible choices which are out there today. Thus the first part of the design will cover this particular subject.

## 3.1. Speech recongnition

Speech recognition is the branch of computational linguistics which enables the recognition and translation of spoken language into text by computers using methodologies and technologies. Also, known as “automatic speech recognition”, “computer speech recognition” or “speech to text”, speech recognition is a vast field of study which incorporates knowledge and research in linguistics, computer science and electrical engineering.

Research in speech recognition has a history which spans decades and has had several waves of major innovations. The most recent wave of such type has been caused by advances in deep learning and big data. This wave of innovations based on deep learning and big data is evident not only because of the large number of academic papers written on the topic, but also by the wide industry adoption and deployment of speech recognition systems based on deep learning and big data. Companies like Google, Microsoft, IBM, Baidu, Apple, Amazon, Nuance, Sound Hound, and many others have made it public that the core technology in their speech recognition systems is based on deep learning.

However, speech recognition has not always been the way it is today. Much of the progress in the field is owed to the rapidly increasing capabilities of computers. In 1971, DARPA (the Defense Advanced Research Projects Agency) funded five years of speech recognition research through its Speech Understanding Program. At the end of the DARPA program in 1976, the best computer available to the researchers was the PDP-10, which featured, among others, 4 MB of RAM memory. Using such computers meant that it would take up to 100 minutes to decode as little as 30 seconds of speech. As years went by and computers became more capable, speech recognition researches began tackling harder problems such as larger vocabularies (one of the end goals of DARPA’s program was a minimum vocabulary size of 1.000 words; a native speaker adult’s vocabulary is 20.000-35.000 words), speaker independence, noisy environments, and conversational speech. Speaker independence, in particular, was a difficult obstacle to overcome. Early speech recognition programs were speaker dependent, which meant that the program first had to be “trained” on a certain speaker so that it would become accustomed to the speaker’s characteristics, such as accent, pronunciation, articulation, roughness, nasality, pitch, volume, and speed. Progress was made on speaker independence first by training on a larger variety of speakers and then by doing explicit speaker adaptation during decoding.

In 1992 DARPA held an evaluation of speech recognition systems. The system which had the best performance was Xuedong Huang’s Sphinx-II. The Sphinx-II system was the first to do speaker-independent, large vocabulary, continuous speech recognition. Huang went on to found the speech recognition group at Microsoft in 1993.

“The 1990s saw the first introduction of commercially successful speech recognition technologies. By this point, the vocabulary of the typical commercial speech recognition system was larger than the average human vocabulary. In 2000, Lernout & Hauspie acquired Dragon Systems and was an industry leader until an accounting scandal brought an end to the company in 2001. The L&H speech technology was bought by Scan Soft which became Nuance in 2005. Apple originally licensed software from Nuance to provide speech recognition capability to its digital assistant Siri.

In the 2000s DARPA sponsored two speech recognition programs: Effective Affordable Reusable Speech-to-Text (EARS) in 2002 and Global Autonomous Language Exploitation (GALE). Four teams participated in the EARS program: IBM, a team led by BBN with LIMSI and Univ. of Pittsburgh, Cambridge University, and a team composed of ISCI, SRI and University of Washington. The GALE program focused on Arabic and Mandarin broadcast news speech. Google's first effort at speech recognition came in 2007 after hiring some researchers from Nuance. The first product was GOOG-411, a telephone based directory service. The recordings from GOOG-411 produced valuable data that helped Google improve their recognition systems. Google voice search is now supported in over 30 languages.”

### 3.1.1. Models, methods, and algorithms

Both acoustic modeling and language modeling are important parts of modern statistically-based speech recognition algorithms. Hidden Markov models (HMMs) are widely used in many systems.

Acoustic modelling

Acoustic Modeling is the process of taking a waveform of speech and analyzing it using statistical models. The most common method for this is Hidden Markov Modeling, which is used in pronunciation modeling to break speech down into component parts called *phones*. Microsoft has been a leading researcher in this field for many years.

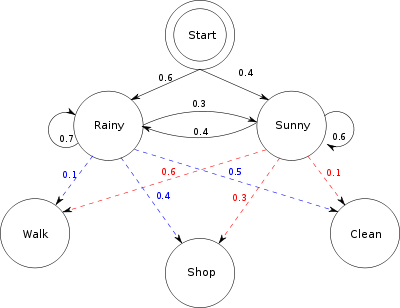
Hidden Markov models

Hidden Markov Modeling is a predictive mathematical model where the current state is determined by analyzing the output. The following example aims to clarify how Hidden Markov models work:

“Consider two friends, Alice and Bob, who live far apart from each other and who talk together daily over the telephone about what they did that day. Bob is only interested in three activities: walking in the park, shopping, and cleaning his apartment. The choice of what to do is determined exclusively by the weather on a given day. Alice has no definite information about the weather where Bob lives, but she knows general trends. Based on what Bob tells her he did each day, Alice tries to guess what the weather must have been like.

Alice believes that the weather operates as a discrete Markov chain. There are two states, "Rainy" and "Sunny", but she cannot observe them directly, that is, they are hidden from her. On each day, there is a certain chance that Bob will perform one of the following activities, depending on the weather: "walk", "shop", or "clean". Since Bob tells Alice about his activities, those are the observations. The entire system is that of a hidden Markov model (HMM).

Alice knows the general weather trends in the area, and what Bob likes to do on average. In other words, the parameters of the HMM are known.”



HMMs work by checking sounds against most probable sounds that come after them. When the pattern matches up correctly, the whole word is detected. For instance, after the sound “th”, the system will check against sounds “e”, “at”, and so forth.

One of the main reasons why HMMs are popular is because they can be trained automatically and are simple and computationally feasible to use.

Modern speech recognition systems use various combinations of several standard techniques to improve results over the basic approach described above. A typical large-vocabulary system would need context dependency for the phonemes (so phonemes with different left and right context have different realizations as HMM states); it would use cepstral normalization to normalize for different speaker and recording conditions; for further speaker normalization, it might use vocal tract length normalization (VTLN) for male-female normalization and maximum likelihood linear regression (MLLR) for more general speaker adaptation.

Language modeling

While Acoustic Modelling is the starting point for speech recognition, it still does not account for several things, such as homonyms and regional variations in pronunciation. That is why Language Modeling is also used. Google has done a lot of research in this area, mainly through the use of N-gram modelling.

When a user interacts with a Google service which makes use of speech recognition, Google makes use of its massive bank of Voice Search and YouTube transcriptions. Google also uses information gathered through their GOOG-411 initiative in order to further improve their accuracy.

This language collection effort resulted in a vast array of pronunciations and dialects, which allowed Google to create a robust dictionary of words and how they sound. This allows for matches that have a greatly reduced error rate than brute force matching based on raw probabilities.

“While Google is a leader in this field, there are other mathematical models being developed, including continuous space models and positional language models, which are more advanced techniques born from research in artificial intelligence. These methods are based on replicating the sort of reasoning humans do when listening to each other. These are much more advanced both in terms of the tech behind them, but difficulty in mapping out these models is also far higher.”

N-gram modelling

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words, or base pairs per the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles. N-gram Modeling works based on probabilities, but it creates a branching tree of possibilities using an existing dictionary of words. Thus, N-gram Modeling eliminates much of the uncertainty in the Hidden Markov Modeling.

As noted above, this method’s strength comes from having a large dictionary of words and usage, not just primitive sounds. This gives the program the ability to tell the difference between homophones, like “beat” and “beet”. N-gram modelling is also contextual, which means that it will know the difference between “beat” and “beet” from the sentence, being able to tell apart a conversation about sports from a conversation about vegetables, for instance.

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Unit | Sample sequence | 1-gram sequence | 2-gram sequence | 3-gram sequence |
| Vernacular name |  |  | unigram | bigram | trigram |
| Computational linguistics | character | …to\_be\_or\_not\_to\_be… | …, t, o, \_, b, e, \_, o, r, \_, n, o, t, \_, t, o, \_, b, e, … | …, to, o\_, \_b, be, e\_, \_o, or, r\_, \_n, no, ot, t\_, \_t, to, o\_, \_b, be, … | …, to\_, o\_b, \_be, be\_, e\_o, \_or, or\_, r\_n, \_no, not, ot\_, t\_t, \_to, to\_, o\_b, \_be, … |
| Computational linguistics | Word | … to be or not to be … | …, to, be, or, not, to, be, … | …, to be, be or, or not, not to, to be, … | …, to be or, be or not, or not to, not to be, … |

### 3.1.2. Other methods and algorithms used in speech recognition

Dynamic time warping

“Dynamic time warping is an approach that was historically used for speech recognition but has now largely been displaced by the more successful HMM-based approach.

Dynamic time warping is an algorithm for measuring similarity between two sequences that may vary in time or speed. For instance, similarities in walking patterns would be detected, even if in one video the person was walking slowly and if in another he or she were walking more quickly, or even if there were accelerations and deceleration during the course of one observation. DTW has been applied to video, audio, and graphics – indeed, any data that can be turned into a linear representation can be analyzed with DTW.

A well-known application of DTW has been automatic speech recognition, where it is particularly useful when comparing speech recording with different speaking speeds. In general, it is a method that allows a computer to find an optimal match between two given sequences with certain restrictions. That is, the sequences are "warped" non-linearly to match each other. This sequence alignment method is often used in the context of hidden Markov models.”

Neural networks

“Neural networks were devised as an acoustic modeling approach in ASR in the late 1980s. Since then, neural networks have found uses in several problems of speech recognition such as phoneme classification, isolated word recognition, and speaker adaptation.

Compared to HMMs, neural networks make no assumptions about feature statistical properties and have several qualities making them attractive recognition models for speech recognition. However, despite their effectiveness in classifying short-time units such as individual phones and isolated words, neural networks are rarely successful for continuous recognition tasks, largely because of their lack of ability to model temporal dependencies.

However, recently LSTM Recurrent Neural Networks (RNNs)and Time Delay Neural Networks(TDNN's) have been used which have been shown to be able to identify latent temporal dependencies and use this information to perform the task of speech recognition.”

Deep Feedforward and Recurrent Neural Networks

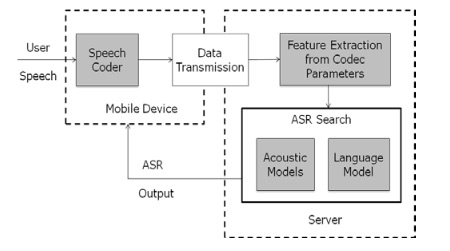
A deep feedforward neural network (DNN) is an artificial neural network with numerous layers of units hidden between the input and output layers. Not unlike shallow neural networks, DNNs can model complex non-linear relationships. DNN architectures generate compositional models, where supplementary layers enable composition of features from lower layers, giving a huge learning capacity and thus the potential of modeling complex patterns of speech data.

It is in 2010 that DNNs have been proven successful in large vocabulary speech recognition by industrial researchers, in collaboration with academic researchers. In this case, large output layers of the DNN based on context dependent HMM states constructed by decision trees were adopted.

### 3.1.3. The necessity of a network connection

Probably the most notable examples today of speech recognition are the intelligent personal assistants offered by Microsoft(Cortana), Apple(Siri) and Amazon(Alexa). And anyone who has used Cortana, Siri or Alexa with a slow internet connection knows that it suddenly becomes a very frustrating experience. That is because the commands sent to Siri are sent over the network to be decoded by Apple, the commands sent to Cortana are sent to Microsoft to be decoded, and commands sent to Alexa are sent to Amazon to be decoded.

The reason these companies have decided to use off-site servers for decoding is that while the decoding could be done on mobile phones or tablets, the sheer amount of processing to be done would mean that delays might be introduced and the battery life of the respective mobile phones or tablets would plummet.



There are, however, desktop models, like Nuance, which use local resources due to the much more powerful hardware and reduced, if not nonexistent, power consumption limitations.

Despite this, offline speech recognition on mobile device is starting to become more widely spread, as Android now allows developers to include offline speech recognition in their apps. Other platforms are also expected to allow developers to make use of offline speech recognition as hardware becomes more powerful.

!!! Here we need to make a connection with what follows (not quite sure how, but maybe Rares does)!!!

## 3.2. Hardware and software choices

!!!We need to add something about the Google Speech API as an alternative to the Echo!!!

The idea behind all the technology which had to be used has been well defined from the beginning of the project at *STIBO* together with the *STIBO Supervisor*, but the market offered numerous hardware components and technologies which the team had to research in order to be able to make the best choices. For the purpose of keeping the research to a minimum a small list of what features should the hardware be able to support, the team made a small list of hardware requirements:

1. The hardware/software must be capable of voice service interaction.
2. The hardware must be capable of hosting a server.
3. The hardware must be capable of displaying video output.

As devices for voice service interaction-capable hardware we had to choose between the *Amazon Echo* and Google Home.

C:\Users\rares\Downloads\voice_interaction (2).png

Figure 1. Voice service interaction-capable hardware

The *Amazon Echo* is a smart speaker developed by *Amazon.com*. The device consists of a 9.25-inch (23.5 cm) tall cylinder speaker with a seven-piece microphone array. The device responds to the name "Alexa". This "wake word" can be changed by the user to either "Amazon" or "Echo".

The device is capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audiobooks, and providing weather, traffic, and other real time information. It can also control several smart devices using itself as a home automation hub [[2](#_References)].

There is currently only one more device on the market which is targeted to the same group of customers the *Echo* is, and performs the same tasks: Google Home. Google Home is a smart speaker developed by Google and supports commands and features similar to the *Amazon Echo*, like streaming through Google Play Music, YouTube Music, Spotify, TuneIn, Pandora, and iHeartRadio. Google Assistant, Google’s intelligent personal assistant, will be included as the main assistant in the operating system of Google Home.

Some of the advantages Google Home has on its side are that it can pull information from various Google accounts and apps, will answer follow-up questions, can communicate with the Google Chromecast, Google-powered speakers, and TVs out of the box, and it is better suited for use in multi-room environments [[3](#_References)].

However, *Amazon Echo* has far wider product integration and features in excess of 3000 Skills (which are third-party services such as Philips Hue or Uber). The most important advantage *Amazon Echo* has over Google Home is that anyone who owns an Amazon Developer Account (free to create) can create and publish a Skill. This is crucial for developers as it allows anyone to build functionality on top of *Amazon’s* *Alexa*. Another very important advantage *Amazon Echo* has over Google Home is that *Amazon* has made *Alexa* available to developers to use in their own creations. As such, the *Amazon* ecosystem is an obvious choice for companies looking to design their own hardware or for DIY-ers wanting to experiment.

Besides these two hardware components, there are also a series of software made for the purpose of human to machine voice interaction into which the team researched. The ones which they considered taking into consideration will be presented below, as well as the motivation and reasoning behind their choices.

1. The Cloud Speech API

This API is still a BETA project made by Google which allows developers to make speech to text conversion by using powerful neural network models in an API capable of recognizing over 80 languages and variants.  
It has various use cases from which some that could be of great use to this project such as:

a) Automatic Speech Recognition or ASR which is powered by deep learning neural networking made to support applications like voice search or speech transcription.

b) Word Hints allows speech recognition to be customized. This is done by providing keywords or phrases that are likely to be spoken in a specific context.

c) Streaming Recognition provides the user with the recognition results while the user is still speaking.

d) Global Vocabulary which has a great vocabulary of over 80 languages from which to recognize.

e) Inappropriate Content Filtering which comes in very useful in case of parental control or to avoid inappropriate language harassment. As this API is not yet complete this feature is only available in some languages.

f) Real-time or Pre-recorded Audio Support allows audio to be captured by an application’s microphone or send from a pre-recorded audio file. It supports multiple audio encodings including FLAC, AMR, PCMU and Linear-16.

g) Noise Robustness takes care of extra noisy audio from different environments without the addition of extra noise cancellation.

h) Integrated API allows all audio files to be uploaded in a request or integrated with Google Cloud Storage.

Unfortunately, even with all these great features from August 2016, Cloud Speech API has stared to be priced per 15 seconds. There is a 60-minute free trial available for trying it out and getting a good feeling of it. The monthly usage is capped at 1 million minutes per month. Also, an approval was needed to use Speech API on embedded devices such as cars, TV’s appliances, or speakers.

2. WIT.AI

A short description of what this WIT.AI is and how could it help this project. Why did we consider researching into it?

3. Web Speech API

A short description of what this API is and how could it help this project. Why did we consider researching into it?

For hardware, capable of hosting a server we had to choose between the Intel Compute Stick, the *RaspberryPI* and the Beagle Bone Black.

C:\Users\rares\Downloads\server.png

Figure 2. Server hosting-capable hardware

The *RaspberryPI* is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries. The project makes use of the *Raspberry’s* latest iteration, the Raspberry Pi 3 [[4](#_References)].

The *RaspberryPI* is by no means the only single-board computer on the market. One of the most well-known alternatives is the Beagle Bone Black, with which the team has previous experience. However, The Beagle Bone Black is unsuited for the project due to its lack of a standardized video output, such as the Raspberry’s integrated HDMI-out, and its lack of a GPU [[5](#_References)]. Both the GPU and the video output are crucial features for this project. The *RaspberryPI* also has the advantage of being one of the most popular single-board computers in the world, and also has a very large community. Thus, using the *RaspberryPI* as a coding base is a more logical choice since help can more easily be found and many more options are available when implementing functions since many more toolchains have been previously compiled for the *RaspberryPI* by the community.

Another option considered by the team is the Intel Compute Stick. The Intel Compute Stick is a single-board computer developed by Intel, and is designed to be smaller than other small-form-factor PCs, while keeping comparable performance. The Intel Compute Stick has been considered for the project due to its small physical footprint, out-of-the-box HDMI output, and the capability of running either Windows-based or Linux-based operating systems [[6](#_References)].

However, the Intel Compute Stick is both more expensive than the *RaspberryPI* and significantly more difficult to modify due to the different design (the RaspberryPI is expressly designed to be built on, while the Intel Compute Stick is a closed system from a hardware point of view). Despite this, the Intel Compute Stick can still be used for the project and would probably perform better than the RaspberryPI due to its more capable hardware, but the team has chosen not to use it because of price limitations. The situation might be different in the future, should a second iteration of the project be considered.

The hardware necessary to display video output was chosen to be a generic HDMI screen. Thus, any screen with an HDMI connection is sufficient.

The items chosen for the system are illustrated below. More technical and detailed information about each hardware component can be found in Appendix B.

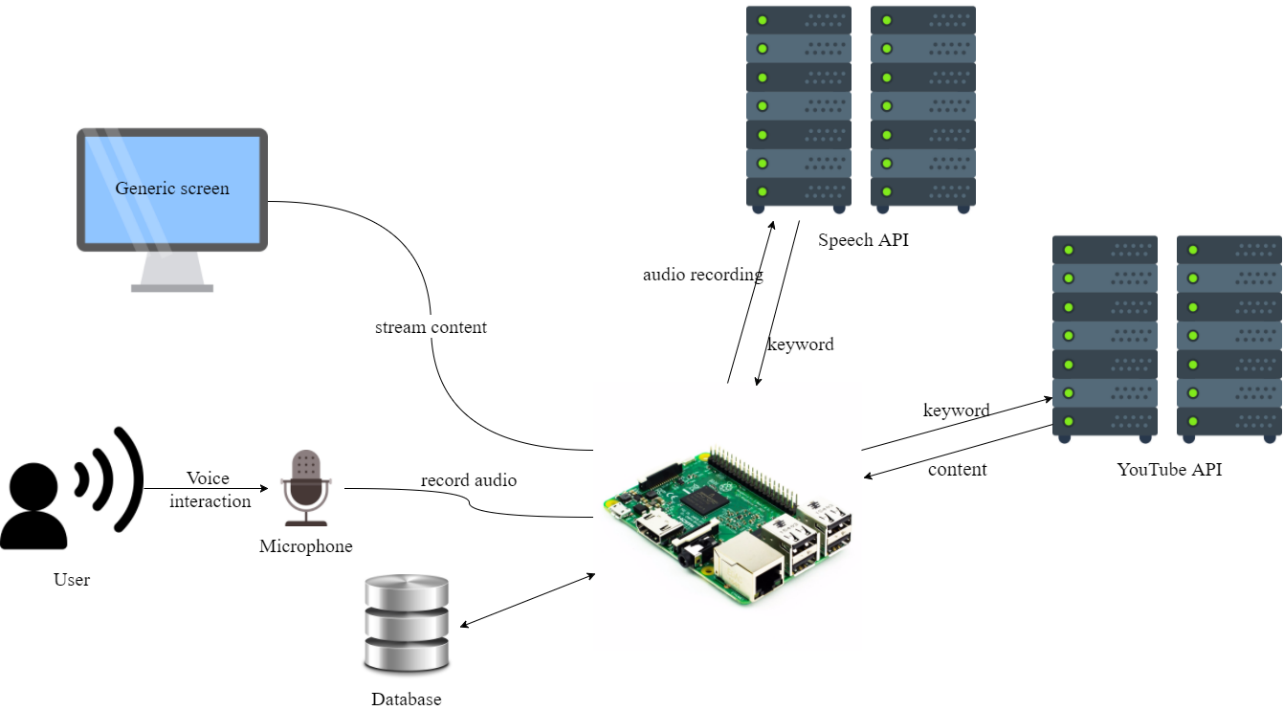


Figure 3. Final system choice

## 3.3. Web Speech API Approach

This design sub-chapter presents the project build using the Web Speech API from Google approach. This is the approach that provided the team with the perfect solution for their needs, but of course as it could be seen in the previous subchapter, there are a lot of technologies from which to choose from, all of them having their own advantages/disadvantages. This approach was found after the team reach several dead ends with some of these approaches. For those interested in more technical details, a full description is found in Appendix F – “Trials”. For those interested in the process that took place during this time please check the Process Report chapter “…”. (I don’t know where will we describe it yet, but it’s probably going to point to the new meetings which we had lately. Also, after we change some parts of the process it will probably come natural).

The image below represents the core system and shows how the components are connect one to another and how the arrows dictate the workflow.



By relating with the Abstract Class Diagram presented in the Analysis Chapter a decision for each part of the system has been made in terms of technologies, hardware, and the communication between these, to reach the goal of the project.

As it can be seen in Figure … the candidate for handling Voice Interaction is a Web Speech API, which converts the user’s speech to text. The command will then be processed on the server which will then send execute the commands and post them on a Single Web Page Application. At that point, the Raspberry PI which has a connection to a screen trough HDMI, is responsible for streaming the output to the user.

### 3.1.1. System architecture

!!! Pavel just point me into the right direction !!!

Here we must present the 3-layer or 5-layer logical Architecture that we are using.

Picture here with the 5-layer Architecture

Presentation – Application – Business Logic – Data Access – Data and Storage Management. Give overall explanation about what it is, how and why do we use it, the motivation behind it.

### 3.1.2. API’s

An Application Programming Interface or API is defined in programming as a set of protocols definitions and tools meant for building software applications. It can be described as a way of communication between various software components.  
In general, by providing the developer with all the building blocks which can be put together to create a program, the API should provide an easier way across.

There a few types of API’s each build for their own purpose. These API’s are:

* Web-based systems
* Operating systems
* Database systems
* Software library
* Computer hardware

Most often the API specifications are made of variables, object classes, data structures and routines. Normally the documentation for an API is given to facilitate usage.

The purpose of these application programming interfaces is to make it easier for developers to use certain technologies when creating applications. The API does this by covering the implementation and giving the developer only objects or actions which are needed, thus reducing the work load on the developer.

This project makes use of such API’s and these two are explained below.

1. Web Speech API

After seeing in subchapter “3.2. Hardware/Software Choices” why this Web Speech API was a good candidate for this project, more in depth knowledge that stands behind this API can be seen here, as well as some examples.

The aim of the Web Speech API is to allow developers to provide a web browser with speech recognition input and text to speech display output. These are features which are not typically available when using screen-reader software. It can support both server-based and client-based embedded recognition and synthesis.  
The API is also designed to support both brief and continuous speech inputs.

When considering security and privacy there are a few rules which must be met:

* User agents must only start speech input session with explicit, informed user consent. The user consent can be for example:
* Users clicks of a visible speech input element which has obvious graphical representation showing that it will start the speech input.
* Give consent to always grant permission to allow speech input for that web page.
* User agents must give the user an obvious indication when audio is being recorded.
* The user agent may also give the user an explanation when interacting with it for the very first time, to let the user know what is going on and inform them on how to tune their privacy settings to disable/enable speech recording if required.
* To minimize the chances of users accidentally forgetting that they allowed web pages to record speech without their knowledge, implementations must abort an active speech input session in case the user lost focus of that web page to another window or another tab.

There are two things which must be taken into consideration before implementing this API:

* The choice of a spoken password input can become problematic from some security perspectives, but it is up to the user to decide if they wish to speak their password.
* Speech input can be potentially used to eavesdrop on users. There are malicious webpages that could trick the user to believe that the speech recording has stopped, when it is still recording.

Speech Recognition Attributes:

The Web Speech API comes with a set of attributes that allow the developer access to various options trough them such as:

* **lang** attribute will set the language of the recognition for the request, using BCP 47 language tag.
* **continuous** attribute which can be set to false in which case the user agent must return no more than one final result in response to starting recognition. It can alsobe set to true, in which case the user agent must return zero or more final results representing multiple consecutive recognitions in response to starting recognition, like a dictation for example.
* **interimResults** attribute controls wheter interim results are returned. If set to true interim results should be returned, otherwise if set to false will not return the results.

Speech Recognition Methods:

Besides the attributes shown earlier the API also has a set of methods meant to help the developers use the Speech Recognition. These can be seen below:

* **start** method – when calling this method, the web application can begin recognition.
* **stop** method – this represent an instruction to the recognition to the recognition service to stop listening to more audio, and try to come up with a result using the audio received up until this point.
* **abort** method – this represents a request that will immediately stop listening and stop recognizing and do not return any other information except that the system is done recognizing.

Speech Recognition Events:

This API uses DOM Level 2 Event Model for speech recognition events. Here are some examples of these events:

* **audiostart** event- is fired when the user agent started the audio capture.
* **audioend** event- is fired when the user agent has finished capturing audio. The **audiostart** event must always be fired before **audioend**.
* **soundstart** event – is fired when sound, possibly speech is detected.
* **soundend** event – is fired when sound is no longer detected. The **soundstart** event must always be fired before **soundend**.
* **speechstart** event – is fired when the speech used for speech recognition has started.
* **speechend** event – is fired when the speech used for speech recognition has ended. The **speechstart** event must always be fired before **speechend**.
* **result** event- is fired when a result is return from the speech recognizer. The **audiostart** event must be fired before
* **start** event – is fired when audio is beginning to be recognized by the system.
* **end** event- is firedwhen the service is disconnected.
* **error** event – is fired when an error occurs during speech recognition.

Speech Recognition Error:

The error event has an interface called the Speech Recognition Error event. It comes with numerous attributes which indicate what went wrong. It is especially useful when debugging and here is a list which will explain what each of them mean:

* **no-speech** – this occurs when no speech was detected.
* **aborted** – this occurs when the speech input has been aborted such as canceling the user speech input from the website.
* **audio**-**capture** – this occurs when the audio failed to be captured.
* **network** – this occurs in the case of a network communication failure.
* **not-allowed** – this occurs when the user agent is not allowing the speech input to initialize for security or privacy reasons.
* **service-not-allowed** – this occurs when the user agent does not allow the web application to request this speech service, but it could allow other services.
* **bad-grammar** – this occurs when the either there was a grammar or semantic tag error, or when the tags are unsupported.
* **language-not-supported** – this occurs when the choose language is not supported.

Speech Recognition Result:

This represents an object of a single one-shot recognition match, either one small part of a continuous recognition or as the complete return result for non-continuous recognition.

It has the following two attributes:

* **length** attribute – represents how many best alternatives are in the item array
* **isFinal** attribute – if the boolean is set to true, then this will represent the final form that the speech recognition will return. If the values is false on the other hand, then this will represent an interim result that could still suffer further changes.

Speech Recognition List:

This object contains a sequence of results given by the speech recognition which represent the complete return set of a continuous recognition. In the case of a non-continuous recognition model, it will only hold a single value.

* **length** attribute – represents the number of results present in the item array.
* **Item getter** – this will return a result from the index into an array of result values. In case of the index being greater or equal to the length, then this will return a null value. It is the user agent’s job to ensure that the length attribute is set to the number of elements in the array.

This concludes the Web Speech API introduction. (need something better, but I didn’t have inspiration).

2. YouTube API

This API is needed, regardless of the design approach and hardware/software choices, as using YouTube for the project’s application requires developers to make use the it and as this project was intended for video or music streaming the team considered YouTube a very practical option, as it is very popular and almost any person with internet has used it at least once in their lives.   
It comes with a very nice set of documentations, intended for developers who write applications that interact with YouTube such as this project.  
It explains the basic concepts of YouTube, the API itself and an overview of the different functions that the API supports. Some of these will be presented here in order to give a better understanding of how the team made use this API.

Now, before anyone can start using the YouTube API there a few steps which need to be followed:

a) A developer needs to have or make a Google Account in order to access the Google Developers Console. Only then can he request an API key which will be used to register the application.

b) In order to submit API requests a project must be created in the Google Developers Console and obtain authorization credentials.

c) After the project has been created, a developer needs to specify in the application that it is registered to use YouTube Data API v3.

d) In case the application uses API methods that require user authentication, OAuth 2.0 authorization is the choice to go for.

Now that everything is set in order to use the API, it is time to take a look at the resources and resource types.

A resource can be described as an individual data entity with a unique identifier. In the table below there is a description for each different data type of resource that a developer can interact with using this API.

Resources

|  |  |
| --- | --- |
| activity | Contains information about an action that a particular user has taken on the YouTube site. User actions that are reported in activity feed include rating a video, sharing a video, marking a video as a favorite. |
| channel | Contains information about a single YouTube channel. |
| channelBanner | Identifies the URL to use in order to set a newly uploaded image as the banner image for a channel. |
| channelSelection | Contains information about a set of videos that a channel has chosen to feature. For example, a section could feature a channel’s latest uploads, most popular uploads, or videos from one or more playlists. |
| guideCategory | Identifies a category that YouTube associates with channels based on their content or other indicators, such as popularity. Guide categories seek to organize channels in a way that makes it easier for YouTube users to find the content which they are looking for. |
| i18nLanguage | Identifies an application language that the YouTube website supports. The application language can also be referred to as a UI language. |
| i18nRegion | Identifies a geographic area that a YouTube user can select as the preferred content region. The content region can also be referred to as a content locale. |
| playlist | Represents a single YouTube playlist. A playlist is a collection of videos that can be viewed sequentially and shared with other users. |
| playlistItem | Identifies a resource, such as a video, that is part of a playlist. The playlistItem resource also contains details that explain how the included resource is used in the playlist. |
| search result | Contains information about a YouTube video, channel, or playlist that matches the search parameters specified in an API request. Although a search result points to a uniquely identifiable resource, like a video, it does not have its own persistent data. |
| subscription | Contains information about a YouTube user subscription. A subscription notifies a user when new videos are added to a channel or when another user takes one of several actions on YouTube, such as uploading, rating or commenting a video. |
| thumbnail | Identifies thumbnail images associated with a resource. |
| video | Represents a single YouTube video |
| videoCategory | Identifies a category that has been or could be associated with uploaded videos. |
| watermark | Identifies an image that displays during playbacks of a specified channel’s video. |

The next table below represent a list of the most common methods that the API supports.

Methods

|  |  |
| --- | --- |
| list | Retrieves (GET) a list of zero or more resources. |
| insert | Create (POST) a new resource. |
| update | Modifies (PUT) an existing resource to reflect data in the request. |
| delete | Removes (DELETE) a specific resource. |

After seeing what resources can be used the next step is to have a look at the supported operations which developers can use.  
The API currently supports methods to list each of the supported resource types, and the table below identifies the operations that are supported for different types of resources. The cells filled with a green OK represent the operations which can be used, and the ones marked with red X represent the operations which are denied.

Supported operations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | list | insert | update | delete |
| activity | OK | OK | X | X |
| channel | OK | X | X | X |
| channelBanner | X | OK | X | X |
| channelSelection | OK | OK | OK | OK |
| guideCategory | OK | X | X | X |
| i18nLanguage | OK | X | X | X |
| i18nRegion | OK | X | X | X |
| playlist | OK | OK | OK | OK |
| playlistItem | OK | OK | OK | OK |
| search result | OK | X | X | X |
| subscription | OK | X | X | X |
| thumbnail | X | X | X | X |
| video | OK | OK | OK | OK |
| videoCategory | OK | X | X | X |
| watermark | X | X | X | X |

In order to make sure that all developers who use the YouTube Data API services do not create applications that unfairly decrees service quality or limit access for other users, Google made a quota counter for each operation. All the requests made with this API including the invalid ones, incur at least one-point quota cost. The quota available to each developer’s application can be found in the Developers Console.

Projects such as this that enable the YouTube Data API will have a default of 1 million units per day, which Google considers sufficient for most API users.

To calculate quota usage Google attributes a cost to each request, which will not be the same for all requests as some can have a bigger data pull then others, or none such as an empty request.

There are two major factors that influence the quota cost for a request:

1. Different types of operations have different quota costs. Some examples can be seen below:

* A simple read operation that just retrieves the ID of the returned resource has a cost of approximately 1 unit.
* A write operation has a cost of approximately 50 units.
* A video upload has a cost of approximately 1600 units.

2. Read and write operations use different amounts of quotas depending on the number of resource parts that each request retrieves. An insert or update operations that write data and also return a resource.   
As an example, inserting a playlist has a quota cost of 50 units for the write operation plus the cost of the returned playlist resource.

Having these rules in mind a developer can try and estimate the number of read, write, or upload requests that the application will send per day, thus ensuring that it will not exceed the limitation of one million units per day.

As it can be seen, a total of one million units per day is actually sufficient for developing a simple application that uses the YouTube Data API. Also in case the quota limit is reached any developer can request additional quota in the Developers Console in the Quotas tab.

In order to ensure that the API uses network, CPU, and memory resources as efficiently as possible, it requires the retrieval of partial resources so that applications avoid transferring, parsing, and storing unnecessary data. Two request parameters allow a developer to identify the resource properties that the API response should include.

1. The “part” parameter which identifies groups of properties that should be returned for a resource.

The API requires the “part” parameter for any requests that retrieve or return resources. The parameter can identify one or more resource properties that should be included in the response. Thus, using this parameter requires a developer to choose the resource components which the application will actually use.

The “video” resource for example has the following parts:

|  |  |
| --- | --- |
| Snippet | contentDetails |
| fileDetails | player |
| processingDetails | recordingDetails |
| statistics | status |
| suggestions | topicDetails |

These parts seen in the table above are objects that contains groups of metadata that the API server will or will not retrieve depending on the request or return which will be made.

This servers for multiple purposes such as:

* Allows API quota management. By increasing the number of parts retrieved by the API response, the API usage will increase accordingly thus decreasing the daily allocated quota amount.
* By not allowing the server to retrieve metadata fields which an application does not use will resolve in reduced latency and less stress on retrieves.
* By reducing the amount of data which a developer does not use in his application will result in reduced bandwidth usage. This could also work the other way around by retrieving all parts of data.

2. The “fields” parameter which filters the API response to only return specific properties within the requested resource parts.

By using the “fields” parameter a developer can filter the API response in a way that it will only include a specific set of fields. By doing this, nested properties from the API response can be removed thus reducing the bandwidth even more.

The rules for the syntax which are supported by the “fields” parameter can be seen below:

* **fields=a,b** – used to select multiple fields.
* **fields=\*** - used as a wildcard to identify all fields.
* **fields=a(b,c)** – used to specify a group of nested properties that will be included in the API response.
* **fields=a/b** – used to identify a nested property.

Here is an example which shows three possible ways to retrieve a playlist’s item ID, title, and position for every item in the playlist:

* **fields=items/id,playlistItems/snippet/title,playlistItems/snippet/position**
* **fields=items(id,snippet/title,snippet/position)**
* **fields=items(id,snippet(title,position)**

This concludes the YouTube API representation. For more information about how this project uses the YouTube API please check the Implementation chapter where the Use Case “Search YouTube video by keyword” will include this API.

### 3.1.3. MEAN Stack

The name MEAN refers to a collection of JavaScript based technologies used to develop web applications. These can be seen in the list below.

* Mongo DB: a powerful, agile, and scalable NoSQL database
* Express: a node.js web application framework which is flexible and provides a set of features to build single and multi-page web applications.
* AngularJS: an open-source front-end web application framework
* Node.js: an open-source JavaScript runtime environment made to develop applications and server tools.

By having all components of the MEAN stack support programs written in JavaScript, applications can have both the server-side and the client-side written in one language.

There are several variations which can alter the original MEAN stack by changing one or more of its components with other, mostly JavaScript frameworks.  
One example is the MEEN stack which replace the JavaScript MVC framework of Angular with Ember.js.  
Even this project will not make use of the full MEAN stack as the front-end side given by Angular will be replaced by Reacts.

!!! Explain why we used React instead of others (Pavel I will need your help here)

1. React.JS Library

React is a library developed by Facebook which is used for front end. It handles the view layer for a web application and can also work with mobile apps.  
React allows developers to create user interfaces in a declarative, efficient, and flexible way.

!!! Explain more about it.

2. Node.js

Node.js is an open source, cross-platform runtime environment build on Google Chrome’s JavaScript V8 Engine. Its purpose is to help develop server-side and network applications. These applications are written in JavaScript and can run on Microsoft Windows, Linux, and OS X. Node.js also comes with a rich library of various JavaScript modules which makes web application development easier using Noje.js to a bigger extent.

The reasons behind the team’s choice into working with Node.js lie in its features:

* As mentioned earlier Node.js is built on Google Chrome’s V8 JavaScript Engine which give the libraries the possibility to execute code faster.
* By simply outputting the data in chunks the Node.js applications never need to buffer data.
* Traditional servers create a limited number of threads to handle requests. Unlike them, Node.js uses a single threaded model and an event looping mechanism which helps the server respond in a non-blocking way. This gives the server high scalability.
* All API’s which are part of the library are asynchronous.

To further back up this list of features showed above, comes a list of great known companies which developed projects and applications using Node.js. Here are only a few of them: Microsoft, eBay, Yahoo!, and PayPal.

What is V8?

It is Google’s open source JavaScript engine. It has high performance and it is written in C++. It is used in Chrome, Node.js and other embedding applications. The V8 implements ECMAScripts and runs on Windows XP or higher versions, Linux systems that use IA-32, Mac OS X 10.5+.   
It can run as a standalone or can be embedded into C++ applications. It also compiles and executes JavaScript source code, manages object memory allocation and can garbage collect unnecessary objects which it no longer uses or needs.

Also Node.js has proven that to be best suited for applications such as:

* Single Web Page Applications
* JSON API Applications
* Data Streaming Applications
* Data Intensive Real-time Applications
* I/O bound Applications

As this project’s interface is based on a Single Web Page Application, and using JavaScript the team found it by far the best choice.

The following diagram shows some important parts of Node.js which were of great use to this project.

C:\Users\rares\AppData\Local\Microsoft\Windows\INetCacheContent.Word\Node_JS_components.png

!!!This is where the Node.js mini chapter ends. The picture is taken from the official website. I need to remake one in Paint or something. I need to know which ones are we using, or are we using all of them? Pavel, I think you can help me out here!!!

3.Express.JS

Express is a web application framework that is both flexible and minimal, and provides a decent set of features to develop both web and mobile applications.

It has the following features:

* Allows to dynamically render HTML Pages
* Allows to middleware setup which respond to HTTP Requests
* Defines a routing table used to perform different actions based on HTTP Method and URL.

When using Express it should be installed along with the following modules:

* body-parser – a Node.js middleware which handles JSON, Raw, Text and URL encoded form data.
* cookie-parser – used to parse Cookie header and populate cookie requests with an object keyed by the cookie name.
* multer – a Node.js middleware for handling multipart data.
* !!!If there are any more then Pavel, you can help me here!!!!

The Express.js application uses a callback function which has two parameters. These are:

* The request object which is the HTTP request. It has properties for the request query string, parameters, body, HTTP headers.
* The response object which represent the HTTP response that is send when an HTTP request is received.

By printing a request or response object a developer can find information such as cookie, session, URL.

!!! If there is anything else that comes to mind let me know!!!

Why choose Express.JS?

* Pavel will have to help me😝
* Pavel will have to help me😝
* Pavel will have to help me😝

4. MongoDB

When choosing a database, the team had to take into consideration a few aspects. They had to choose between a SQL database which deals with complex queries and transitions and builds relations between entities to ensure integrity of the data, and a NoSQL database which designed to excel in speed and volume but gains these advantages at the cost of losing relations.

By using Node.js, it can talk to and query almost any database solution, but after some research it was discovered that the NoSQL database, MongoDB, is best suited for a low-powered and low-resource device like the Raspberry PI which is used in this project.

MongoDB is an agile and scalable NoSQL database. Compared to the traditional columns and rows of relational database, MongoDB has document store model, in which data objects are stored as separate documents inside a collection. These documents are stored as binary JSON or BSON objects.

The motivation of using the MongoDB language stands behind the possibility to implement a data store that provides high performance, high availability, and automatic scaling.

MongoDB is simple to install and offers great website back-end storage for high-traffic websites that need to store data.

Here are some reasons why MongoDB was chosen for this project and why it is such a popular NoSQL database:

* MongoDB is one of the highest-performing database available.
* The MongoDB replication model give the possibility to maintain scalability while still being able to output high performance.
* MongoDB scales horizontally using sharding. The developer chooses a shard key, which determines how the data in a collection will be distributed. Afterwards, the data is split into ranges and distributed across multiple shards.
* MongoDB can run over multiple servers, balancing the load or duplicating data to keep the system up and running in case of hardware failure.
* MongoDB uses collections also known as capped collections. Using this type of collection, MongoDB maintains insertion order and, once the specified size has been reached, it will start acting like a circular queue.
* MongoDB is document oriented, thus eliminating the need for data to be transferred from rows to objects and back.
* JavaScript can and is used in queries, aggregation functions, and are sent directly to the database to be executed.
* Because objects are stored as objects and not by using SQL strings, MongoDB is made to be not susceptible to SQL injection.

As mentioned in the list above MongoDB groups all its data with the use of collections. This collection can be seen as a simple grouping of documents that have the same or a close similar purpose. A collection can be considered as a table in a traditional SQL database but with one major difference which is, that a collection is not forced to follow a strict schema. This reduces the need to break a document’s items apart into more different tables, as the practice goes in SQL implementations.

The document represents a single entity of data in MongoDB. They can contain embedded subdocuments that provide the application with an inherent data model. These documents are stored as BSON, which is a lightweight binary form of a JSON.

!!! Show an example when we have one:

### 3.1.5. Git & GitHub

Rares

# 4. Graphfical User Interface

Rares and me

# 5. Implementation

Pavel’s time to shine

# 6. Testing

Rares and me

# 7. Results

Rares and me

# 8. Discussion

All three of us

# 9. Improvments

All three of us

# 10. Conclusion

All three of us