Verbal Communication Application: Using IBM's Watson and Node-RED to Aid the Communicationally Challenged

Norah Abanumay, Babatunde Adeola, Yiming Fan, Ryan Finlayson, Xiaofeng Fu, Kristen Hayzelden, Assia Moujdi

Abstract

Speech is an essential communication method. However, not all people are capable of verbal communication. These individuals may suffer from injury, brain damage, mental health issues, selective muteness, or a variety of other ailments. The current technologies in place for those who are less capable of verbal communication (text-to-speech, visual charts, etc.) can greatly benefit their users, but have major drawbacks: namely speed. We worked with IBM to develop a progressive web application to generate sentences based on a keyword search. This will benefit users as it will cut down on the time taken for them to voice a thought. Instead of typing a full sentence, they will simply type a keyword and have sentences generated for them. The web application generates the sentences by using IBM's Watson and its Tone Analyzer to filter relevant tweets from Twitter.

0.1 Access our Work

A live version or our tool (though not currently maintained) is available at : https://watsontwittercomm.eu-gb.mybluemix.net/

The code behind the application is also availabe under UCL's open source at: https://github.com/KHayzelden/ibm_communication_tool.git

1 Introduction

1.1 Problem

Verbal communication is one of the main forms in which we communicate daily. However, not all people are capable of verbal communication. Speech and communication difficulties affect 2.5 million people in the UK alone [1].

There are many tools on the market which aim to provide a similar result as the tool we have created. These fall into two main categories: text-to-speech and pre-set sentence selectors.

However, the problem with the existing solutions boils down to speed and limited discussion topics. The text-to-speech tools require users to type in the entirety of what they want to say. This can be especially slow for users whose communication needs stem from a condition that limits their mobility and therefore their typing speed (e.g. stroke, accident, etc.).

The current market solution to the speed issue regarding text-to-speech tools are applications that contain lists of pre-made sentences. These applications will provide users with lists of sentences in categories. For example, a user can select the "food" category then voice the "I am hungry" option. While these tools can speed communication on basic topics, they have limited customization features and are unfit for discussing trending topics (like the news, movies, etc.).

1.2 Aim

Our aim is to find a middle ground and create a tool that allows for conversation on modern topics without requiring the user to type the full sentence they wish to speak. This will reduce the effort and speed for users compared to text-to-speech tools - but will provide them with more conversation options than pre-made sentence tools.

1.3 Solution

Our solution is a progressive web application [2] (a web application offering offline access) that allows a user to input a keyword or short phrase relating to a popular topic on which they have an opinion. These topics can belong to multiple categories such as current events, films, music, sports, etc.

We aimed to create an application that can find a middle ground for users. One that allows users to quickly express complex opinions on trending topics. The users should be able to express their opinion faster than it would take to type the full sentiment. This is done by allowing users to type a keyword or phrase which is then used to query Twitter and obtain relevant trending tweets.

The tweets are then filtered for content (removing expletives, personal information, etc.) and then sent to IBM's Watson [3] Tone Analyzer. The Tone Analyzer returns up to 20 tweets that could convey the opinion of the user. The 20 sentences are ranked by how likely they are to be what the user is going to select, before being presented to the user on their device.

Once the user selects which of the sentences best fits their sentiment, the device voices the sentence for the conversational partner. Users also have access to their full search history and can create bookmarked sentences. All this information is saved in a local database and synced with their online database. Ensuring the maximum communication efficiency for users.

There are some limitations as this is a proof of concept application and is not ready currently for a full launch. These are discussed later.

2 Objectives

As mentioned previously, our application was conceived with one major goal in mind. A solution to faster communication for the communicationally challenged. Specifically, one that would allow for faster communication than text-to-speech tools offer while providing them with more flexibility than pre-generated sentence selection tools.

2.1 Goals

To achieve our overall goal, we have a list of sub goals that this application aims to achieve.

- **G1**: Achieve more natural conversation for users
- **G2**: Increase user response speed when conversing
- **G3**: Reduce time required for user to convey an opinion
- **G4**: Increase the number of conversations held by users
- **G5**: Generate sentences that convey general opinions on popular topics
- **G6**: Provide users with an improved communication tool that they can solely rely on

2.2 Requirements

To meet our goals, we devised a list of requirements of our system to ensure we created the best possible product for our users while meeting our goals. These requirements fit into a variety of categories.

2.2.1 *General Requirements*

- **R1**: Use Twitter as a source of content for different opinions on popular topics
- **R2**: Use Watson's Tone Analyzer to determine the sentiment of Tweets and filter unwanted tweets
- **R3**: Use Node-RED [4] to filter Tweets further for content (such as links)
- **R4**: Use tracking to distinguish the most commonly chosen results to improve result quality
- **R5**: Use a local and online database for maximum offline functionality

2.2.2 *User Requirements*

- **R6**: Allow users to register an account for their information to be available on multiple devices
- **R7**: Allow users to view and edit their profile and account information
- **R8**: Allow users to input a keyword and receive 20 relevant sentences
- **R9**: Allow users to select a sentence and have it voiced for them
- **R10**: Allow users to save keywords and sentences as bookmarks for ease of access
- **R11**: Allow users to select a bookmark and have the appropriate task completed for them
- **R12**: Allow users to type and directly voice a sentence without searching
- **R13**: Allow users to create conversations
- **R14**: Allow users to specify a desired tone or emotion when searching to filter results
- **R15**: Allow users to view their current connectivity status

2.2.3 *Interface Requirements*

- **R16**: The application should have a friendly and simple user interface
- **R17**: The application should provide accessibility features (such as: larger text, higher contrast, highlight links, etc.)

2.2.4 *Database Requirements*

- **R18**: Store users' login credentials
- **R19**: Store users' bookmarks, settings, and keywords
- **R20**: Persist data while the application is offline

2.2.5 *Availability and Resilience Requirements*

- **R21**: The system should be available for as close to "24/7" as possible
- **R22**: The system should provide offline functionally (for all functions that do not rely on a stable internet connection)
- **R23**: The system should be backed up for recovery in case of disaster or corruption

2.2.6 *Quality Requirements*

- **R24**: The application should be available and functional with devices regardless of platform (mobile, tablet, etc.), OS (iOS, Android, Windows), or screen size
- **R25**: The project should provide full documentation for further development or replication
- **R26**: The project should have no associated costs in its initial state (as it is a proof of concept)
- **R27**: The application should be tested to ensure functional quality

2.2.7 *Security Requirements*

- **R28**: The data captured should be protected and not hold any sensitive information
- **R29**: The user account details should be stored securely and not revealed to any other party
- **R30**: The log in process should be fully authenticated

3 Design

3.1 Architecture

As our application is a proof of concept, we needed to ensure our system's architecture would be simple to implement and focus on interchangeability and scalability. For this iteration of the application, scalability was not a major concern, but we understand that if this were to be developed into a full product, scalability would need to be implemented in the core architecture. Interchangeability is also important for the future work in this system as many of the external systems we chose to use are IBM products and other companies may wish to implement this work with their own systems.

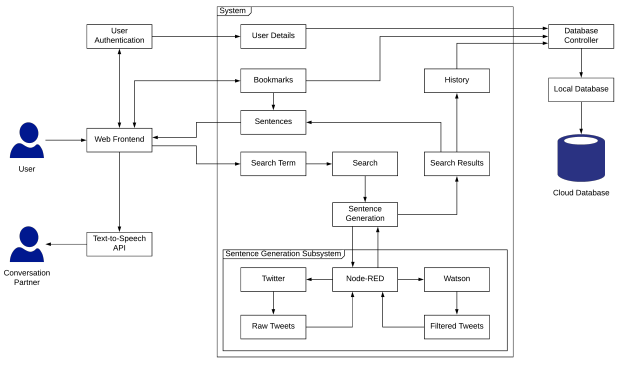

Figure 1: The basic architecture of our system.

Looking at our architecture diagram, there are three main places where we have implemented specific architecture to ensure interchangeability. The web frontend allows us to change the frontend where necessary to suit the needs of the company, the sentence generation layer allows us to change out our sentence generation subsystem (key for a full implementation as Node-RED is not scalable - this is discussed later), and the database controller allows us to interchange the preferred cloud database.

Another key point in our architecture is our implementation of two databases. We have a local database that syncs with a cloud database when possible. This allows us to provide a key functionality requirement: offline functionality. The local database provides users with their data while offline and the cloud database stores their information for use across multiple devices and local data recovery.

3.2 Interface

Our user interface design process was one of the most challenging components of this project. As discussed earlier, the intended userbase for this application is a particularly venerable group with differing ranges of mobility, technology proficiencies, and accessibility requirements. These vague but important requirements have led to large changes in our user interface design over time. These versions have gone through four main stages: initial conception, wireframe, mockup, and final interface.

3.2.1 *Initial Conception*

Our initial plan was to have an application that would have two different interfaces. The type of interface would be chosen by the user to suit their technology proficiency. There would be simplified interface that would only contain searches, bookmarks, and history. Alternatively, there would also be a full interface that would contain search, conversations, new conversation, search history, bookmarks, profile, settings.

3.2.2 *Wireframe*

The initial step was to design a wireframe of the application to receive feedback from involved parties on an initial design. The wireframe was a flat image built using the tool Pencil [5] for the tablet view only. These were then fed back to our clients, IBM, and a charity group, The Ace Centre, who specialize in custom communication technologies for the less able.

3.2.3 *Mockup*

Using the online tool Moqups [6], we then made a fully interactive mockup for our application. This allowed us to test the flow of the application. It was at this stage that we received two critical points of feedback.

(1) The conversations feature proposed in the requirements would be too complicated for the majority of our users
(2) The two different interfaces (simplified and full) could be difficult to implement and confusing for users

This feedback led to a redesign of our interface where we reduced the application to one interface, and we have moved the conversations feature into a reworking stage.

3.2.4 *Final Interface*

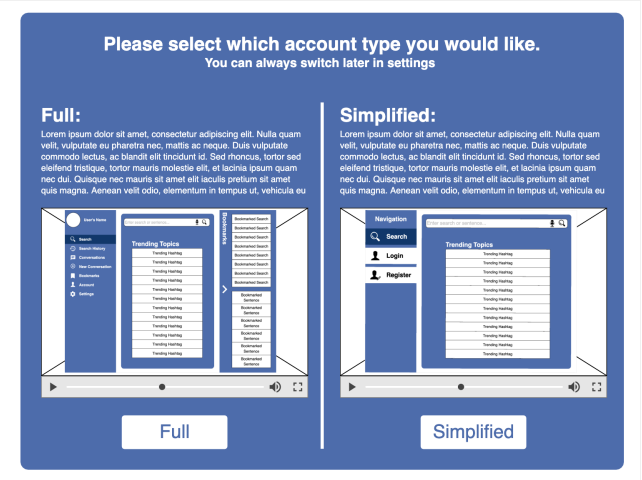

Figure 2: The initial mockup of our registration page where users would be able to choose between two UIs.

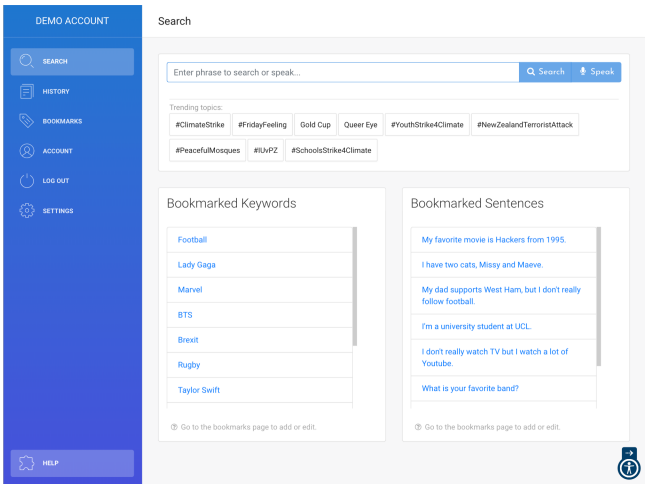

Figure 3: The final user interface for our system.

Our final interface was designed using a mixture of custom CSS and bootstrap templates. We added key features recommended to us as well. For example: a help button in the navigation bar (to ensure users have a guide to our application) and text on all buttons to ensure button function is clearer to our users.

4 Implementation

The principle concept for our implementation is a progressive web application made with node.js. This provides an application that can have offline functionality which is vital for our project as it is a communication tool. Additionally, node.js is a powerful developer tool for fast deployment and simple scalability. To create our application, we used a variety of systems that best suited the need of each component.

4.1 Overview

Our users will interface with a progressive web application that was created with node.js. This application is designed to generate sentences which is carried out by a Node-RED flow that pulls the content from Twitter and filters through Watson's Tone Analyzer. The information is presented to the user, who selects the content they would like voiced for them. The voicing his handled by an open source API: Responsive Voice [7].

For storage, our application uses the PouchDB [8] and Cloudant [9] databases. Core content is cached for offline functionality. Four interface accessibility features the application uses UserWay [10], an accessibility widget.

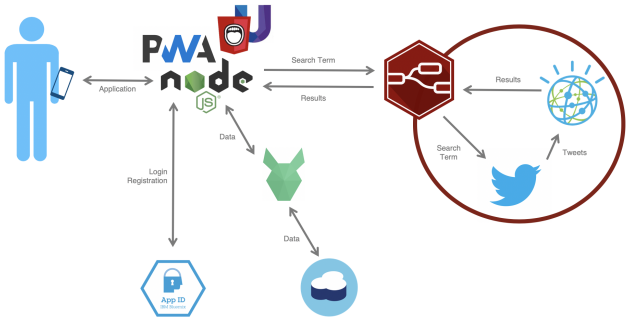

Figure 4: The general flow for our system. This shows the user interacting with a Node.js PWA that uses Userway, Responsive Voice, Node-RED, PouchDB, Cloudant, and AppID.

4.2 Tools and Architecture

4.2.1 *Front End*

The front end of our application was created using a combination of HTML5, JavaScript, Custom CSS, Bootstrap, and EJS.

4.2.2 *Sentence Generation*

The main function of our application allows users to enter a keyword or short phrase from which the application will generate 20 relevant sentences using content from Twitter. To achieve this, we have created a Node-RED flow that will take the keyword and produce our sentences.

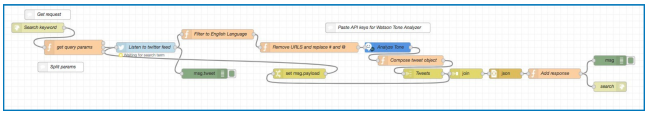

Figure 5: The nodes in our Node-RED flow that is used for our sentence generation.

The process is as follows:

(1) The user enters their desired keyword or phrase.
(2) The application uses Socket.io to send the keyword to the server.
(3) The keyword is passed to the Node-RED flow.
(4) The Node-RED flow uses the keyword to pull a stream of relevant tweets.
(5) The tweets are filtered in Node-RED for content (such as mentions and URLs).
(6) Tweets are filtered to the English language only and transformed from a Twitter post into a sentence string (by removing special characters such as @ and #).
(7) The filtered tweets are sent to Watson's Tone Analyzer to further filter for tone, by providing social, emotional and language type analysis.
(8) Once the flow has found 20 relevant tweets, a blank injection is passed back to the Twitter node to stop the endless flow.
(9) The 20 results are then processed into a json file and sent back to the application.
(10) The application pulls the data using EJS and displays it to the user.

4.2.3 *Data Storage*

Our application uses two data storage systems to achieve both offline functionality and online syncing: PouchDB and IBM's Cloudant.

PouchDB is a database architecture based on the local browser and we have used it to achieve the offline storage for our application. All application data is stored locally and cached using PouchDB. This allows users to have access to their search history and bookmarked sentences for maximum offline functionality.

Cloudant is an "SQL-free" cloud-based database that was ideal for our application due to its simplicity. The database is structured using json files and allows us to quickly synch the PouchDB and local information while connected.

In our initial attempt, the database was server wide. Naturally, this was not viable as all users were sharing a single database and had access to each other's information. This was rectified by separating the users into their own personal databases upon registration.

4.2.4 *User Authentication*

To allow users to create an account, while insuring their credentials were handled securely and efficiently, we chose to use IBM's AppID [11]. AppID provides our application with a secure authentication service that is flexible for future expansion and customization (social media login, custom security standards, etc.).

4.2.5 *Progressive Web Application*

The website was converted into a Progressive Web Application (PWA) and therefore contains features related to both native and hybrid type applications. For instance, users are able to add the application icon directly to the home screen, as they would any native application. Users are also able to access the site within the browser like a regular website.

4.2.6 *Caching Strategy*

The *Stale-While-Revalidate* pattern was chosen as our main caching strategy, to ensure that static assets (such as HTML and CSS code) are available to the user offline. Service workers were responsible for deciding exactly which assets are going to be cached. User requests operate in a cache first manner, as the application will

respond to the request as soon as possible by checking the cached assets first. If the response is not available within the cached assets, the application will fallback to the network to fetch the request and update the cache accordingly.

There are several benefits to caching such as increasing the availability of the application as the application can be used offline, reduced latency and network traffic. These improvements increase user satisfaction.

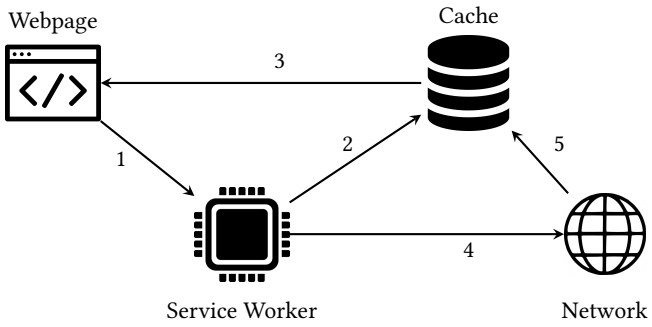

Figure 6: Stale-While-Revalidate Caching Pattern

4.2.7 *General Functionality*

Many of our application's more general components are accomplished using JavaScript, JQuery, Scoket.io, open source APIs, and EJS. They are explained below.

Loading Data
All data displayed to the user is loaded using EJS to pull the information from the local PouchDB database.

History Searching
To prevent unnecessary calls to the database, we have used a combination of JavaScript and JQuery to allow users to search their history and simply hide the content that is not relevant to their search. They can search via keyword or date.

Text-to-Speech
Our text-to-speech functionality is entirely handled by Responsive Voice, an open source JavaScript API with features such as voice gender, tone, and speed.

Bookmarks
Using Socket.io [12] we are able to connect our frontend and backend JavaScript. This allows us to update the database with the user's bookmarks when added, changed, or deleted.

4.2.8 *Testing*

While testing was difficult for us, we were able to achieve some testing measures. For our UI testing, we have applied a "dumb" monkey testing tool [13]. The "dumb" monkey testing ensures that all of our UI is functional without considering any backend of our application. Our tests ensured that we did not have any crashes, broken components, or 404 errors.

For functionality testing we have implemented a CodeceptJS [14] through the Selenium framework [15] which allowed us to create scenario-based test cases. We have made over 20 test cases for each main use case of our system. Naturally, we hesitate to call this fully tested as due to time limitations we have written potentially biased test cases.

We have been unable to carry out any user tests (see limitations).

5 Results

Our application is currently live. As planned, it is a progressive web application using Node.js and offers partial offline functionality. While our project is currently accessible and functional, there are some requirements we were unable to satisfy in this iteration of the project. Overall, we deem the project successful as a proof of concept, since the main goals were satisfied. We would, however, refine and add feature before we would consider this application a viable product.

5.1 Goal Satisfaction

Returning to our initial goals helps identify the success of our project. We are unable to say for fact that we have met our main goals at this moment, but we believe that we have created an application that will satisfy users and achieve our goals. This lack of confirmation on our goals stems from our limitations with user testing. As this product is something made to improve quality of life, we cannot say for certain we have achieved this without user input. We do, however, believe that our application is a stepping stone to such an improvement and with future work involving user testing we could prove this.

5.2 Requirement Satisfaction

While we struggle to factually prove our goal satisfaction, we can compare our results to our initial requirements to ensure the product we designed is of quality and functions as intended.

5.2.1 *General Requirements*

- **R1: Use Twitter as a source of content for different opinions on popular topics**
  Status: Requirement Met
- **R2: Use Watson's Tone Analyzer to determine the sentiment of Tweets and filter unwanted tweets**
  Status: Requirement Met
  Comment: This requirement was met but we have reduced the desired amount of filtering to meet our speed goals
- **R3: Use Node-RED to filter Tweets further for content (such as links)**
  Status: Requirement Met
  Comment: This requirement was met but we have reduced the desired amount of filtering to meet our speed goals
- **R4: Use tracking to distinguish the most commonly chosen results to improve result quality**
  Status: Requirement Not Met
  Comment: This feature was not possible to implement at this stage due to a lack of threading discussed in limitations

- **R5: Use a local and online database for maximum offline functionality**
  Status: Requirement Met

5.2.2 *User Requirements*

- **R6: Allow users to register an account for their information to be available on multiple devices**
  Status: Requirement Met
- **R7: Allow users to view and edit their profile and account information**
  Status: Requirement Partially Met
  Comment: Due to limitations with AppID, users can edit their password but not their email
- **R8: Allow users to input a keyword and receive 20 relevant sentences**
  Status: Requirement Met
- **R9: Allow users to select a sentence and have it voiced for them**
  Status: Requirement Met
- **R10: Allow users to save keywords and sentences as bookmarks for ease of access**
  Status: Requirement Met
- **R11: Allow users to select a bookmark and have the appropriate task completed for them**
  Status: Requirement Met
- **R12: Allow users to type and directly voice a sentence without searching**
  Status: Requirement Met
- **R13: Allow users to create conversations**
  Status: Requirement Not Met
  Comment: This feature was moved into a reworking stage and is discussed in limitations
- **R14: Allow users to specify a desired tone or emotion when searching to filter results**
  Status: Requirement Not Met
  Comment: This feature was dropped as it would cause the search results to take too long due to lack of threading
- **R15: Allow users to view their current connectivity status**
  Status: Requirement Not Met

5.2.3 *Interface Requirements*

- **R16: The application should have a friendly and simple user interface**
  Status: Requirement Met
  Comment: The interface was designed with minimal clutter, large buttons, and detailed descriptions
- **R17: The application should provide accessibility features (such as: larger text, higher contrast, highlight links, etc.)**
  Status: Requirement Met
  Comment: Achieved with the UserWay widget

5.2.4 *Database Requirements*

- **R18: Store users' login credentials**
  Status: Requirement Met
  Comment: While not stored in the database for security, they are held through AppID
- **R19: Store users' bookmarks, settings, and keywords**
  Status: Requirement Met
- **R20: Persist data while the application is offline**
  Status: Requirement Met
  Comment: Achieved with caching and a local PouchDB database system

5.2.5 *Availability and Resilience Requirements*

- **R21: The system should be available for as close to "24/7" as possible**
  Status: Requirement Not Met
  Comment: Due to the quota limitations and this being a proof of concept tool - we, ultimately, did not verify this
- **R22: The system should provide offline functionally (for all functions that do not rely on a stable internet connection)**
  Status: Requirement Met
  Comment: All functions that do not rely on an internet connection work while offline
- **R23: The system should be backed up for recovery in case of disaster or corruption**
  Status: Requirement Met
  Comment: The application data is stored on the cloud through Cloudant

5.2.6 *Quality Requirements*

- **R24: The application should be available and functional with devices regardless of platform (mobile, tablet, etc.), OS (iOS, Android, Windows), or screen size**
  Status: Requirement Met
  Comment: Using bootstrape and responsive CSS we have ensured our application is suitable for all devices
- **R25: The project should provide full documentation for further development or replication**
  Status: Requirement Met
  Comment: In our repository are "read me" files with details on how to replicate the project
- **R26: The project should have no associated costs in its initial state (as it is a proof of concept)**
  Status: Requirement Met
  Comment: While there are limitations due to the free components used, the current state of the project has no associated costs

- **R27: The application should be tested to ensure functional quality**
  Status: Requirement Partially Met

5.2.7 *Security Requirements*

- **R28: The data captured should be protected and not hold any sensitive information**
  Status: Requirement Met
  Comment: All sensitive data is stored through a secure third party
- **R29: The user account details should be stored securely and not revealed to any other party**
  Status: Requirement Met
  Comment: AppID provides a secure login and storage system
- **R30: The log in process should be fully authenticated**
  Status: Requirement Met

Note that this project is a proof of concept and as such we have not considered any GDPR [16] compliancy. Therefore, our application may not be GDPR compliant in its current state.

5.3 Summary

We were able to fully meet nearly all our requirements and the few that we have not been able to meet are easily fixed with additional resources and more time. Moreover, none of the unsatisfied requirements are relevant to the main functionality of our application.

Ultimately, we would consider this project as success as a proof of concept application perfect for future work in the field of alternative communication solutions.

6 Limitations

Due to the nature of our project, we rely on a lot of external services. While this is beneficial in many ways (ex. AppID providing us with a secure user authentication system in no time), we are also limited by these external services. Three main components of our application are under quota restrictions that would need to be addressed for this application to become a viable product.

6.1 Watson

Our application uses IBM's Watson for sentence filtration as part of our Node-RED flow. The IBM accounts we are using for this application are free accounts and come with a limited quota for Watson use. This quota does not refresh over time, meaning that after a certain point our application can no longer filter with Watson's Tone Analyzer without changing accounts or purchasing more quota.

6.2 Node-RED

Our sentence generation system is a Node-RED flow connecting IBM's Watson and Twitter to create the content for our application. This flow is called whenever a user searches a term on our application. The current limitation is that Node-RED is a single threaded service. This means that only one search can be done at a time and any further searches are added to a queue and wait for the first to be completed. This does not allow for scaling. We are looking at moving the Node-RED flow into a thread compatible solution.

6.3 AppID

For our user authentication system, we are currently using IBM's AppID. This, similarly to the Watson situation, has a quota for free use and at a certain point we are unable to log users in or register users without purchasing more quota or changing accounts.

6.4 Response Speed vs Quality

The entire purpose of our application is to speed the communication process for our users. Unfortunately, we have had to balance quality and speed. When using content from Twitter, there are a variety of items to remove in order to emulate a spoken sentence (ex. emojis, expletives, links, mentions, etc.) but the more filters we apply to the Tweet, the longer the process takes. This has limited the quality of the responses we are giving to the users as we have chosen to filter only a portion of what we would prefer to filter.

6.5 Testing

Due to the time limitations and the vulnerable nature of our user base, we have not been able to carry out any accurate user testing. We are in contact with the Ace Centre, a charity group who specialize in communication technologies for the less able, to try and get some users to test our application. Until that is worked out, we only have done "user tests" ourselves. These are better than nothing but are not accurate to our user base.

7 Future Work

The application we have built is a proof of concept which naturally opens to a large variety of future work and directions for this concept. Beyond working on solutions to our limitations, the following features are our ideas for the next stage of this application.

7.1 Conversations

The original draft of our application included the conversations feature we have previously discussed. While it was deemed unfit for our current application, a rework of this feature would be a vital addition to our application. The conversation feature will allow a user to group their history and access it later as a way to have relevant searches in history grouped together and save time.

7.2 More Data Sources

Currently, our application takes content from Twitter to generate the sentence for our users. A step forward would be to pull content from other areas of the web. Sites such as Amazon, Google Reviews, Yelp, Reddit, and Facebook could provide more content to filter and can lead to a wider range of results.

7.3 More Sentence Selection Options

Currently our application only displays twenty sentences for the user to choose from. If the twenty sentences do not contain phrases that the user wishes to say, the user would have to perform another keyword search to reload the responses. As a future work, we can add a setting which allows the user to select how many sentence options they wish to have available to them. This can range from twenty responses minimum to 100 responses maximum, allowing the user to toggle to their desired option. Increasing the sentence

amount counteracts the issue of having to reload responses, by increasing the chances of a sentence appearing that suits what the user wishes to say.

7.4 Speech-to-Text

A speech-to-text component would greatly benefit our application. Keeping a record of the conversation partner's replies to the user would not only help the user better organize their history - but could be evolved to help predict responses and further improve user response speed.

7.5 Detailed User Profiles

The current application has a small set of profile specific features but ideally a more detailed profile and a user settings page would improve our application. Users being able to enter details about themselves (such as their age, gender, location, etc.) could allow the application to predict what topics and replies they may prefer. Additionally, more settings can help the user feel like the application is more personal to them. Namely, the ability to change the text-to-speech gender and speed to better represent the user.

7.6 Custom Sentence Generation

As hinted before, the ability to generate results that are tailored to the user would be a huge step forward for this concept. One way to do that could be social media profile integration. Allowing our users to link their other social media accounts (Amazon, Facebook, Netflix, etc.) would provide data and allow us to suggest more relevant content to our users.

Acknowledgments

We would like to give special thanks John McNamara from IBM for the opportunity to work on this project and guiding us through every stage of this application's development. We also would like to thank Simon Parkinson from the DOT Group and Will Wade from the Ace Centre for their help with our project.

We would also like to thank those at UCL who provided us with guidance and ensured the success of this project:
Dean Mohamedally, Emmanuel Letier, and Eric-Tuan Le.

References

[1] "Private Speech & Language Therapy, Speech Therapists For Schools, Speech & Language Therapists & Autism Assessments âĂŞ Integrated Treatment Services," Integrated Treatment Services. [Online]. Available: http://integratedtreatmentservices.co.uk/. [Accessed: 25-Feb-2019].
[2] "Progressive Web Apps | Web | Google Developers," Google. [Online]. Available: https://developers.google.com/web/progressive-web-apps/.
[3] "IBM Watson," IBM. [Online]. Available: https://www.ibm.com/watson.
[4] "RED," Node. [Online]. Available: https://nodered.org/.
[5] "Pencil Project," Home - Pencil Project. [Online]. Available: https://pencil.evolus.vn/.
[6] "Moqups Âů online mockups made simple," Online Mockup, Wireframe & UI Prototyping Tool Âů Moqups. [Online]. Available: https://moqups.com/.
[7] "ResponsiveVoice Text To Speech," ResponsiveVoice.JS Text to Speech. [Online]. Available: https://responsivevoice.org/.
[8] "The Database that Syncs!," PouchDB, the JavaScript Database that Syncs![Online]. Available: https://pouchdb.com/.
[9] "Cloudant - Overview," IBM. [Online]. Available: https://www.ibm.com/cloud/cloudant.
[10] "Web Accessibility Widget," UserWay. [Online]. Available: https://userway.org/.
[11] "App ID - Overview," IBM. [Online]. Available: https://www.ibm.com/cloud/app-id.
[12] "Socket.IO," Socket.IO, 18-Feb-2019. [Online]. Available: https://socket.io/.
[13] Fofism, "Fully automated monkey testing. Learn more," Monkey Test It: Automated website testing by AI . [Online]. Available: https://monkeytest.it/.
[14] "CodeceptJS Âů Modern End 2 End Testing Framework for NodeJS," CodeceptJS. [Online]. Available: https://codecept.io/.
[15] "Browser Automation," Selenium WebDriver. [Online]. Available: https://www.seleniumhq.org/projects/webdriver/.
[16] "The EU General Data Protection Regulation (GDPR) is the most important change in data privacy regulation in 20 years.," EUGDPR Home Comments. [Online]. Available: https://eugdpr.org/.