6 November 2022

CFG Full-Stack Autumn 2022 Group 5:
Abigail Unwin, Despoina Tsounaka, Justyna Janiszewska, Sirad Bihi, Zeynep Kurugollu

ABLE-SPEAK: THE NON-VERBAL COMMUNICATION APP

I. INTRODUCTION

We understand how frustrating it is for non-verbal communicators to connect with loved ones and those around them, so we've developed the able-speak app to bring people together and help transform their lives. This app was designed not only to help boost confidence and reduce isolation for non-verbal people, but also to give them a voice.

II. BACKGROUND

"Able-speak" is a non-verbal communication application that gives the user the ability to communicate with others through image-to-speech functions. To be as inclusive as possible, we provide images accompanied with text, in case the user has difficulty reading. Selected images are then selected using a drag and drop function, then translated to speech with the push of the microphone button.

This web application is ideally suited to a portable device (phone or tablet) to accommodate the mobility needs of its user, but can also be used on a desktop computer. From market research, we understand that the PECS (Picture Exchange Communication System) is a popular method for enabling non-verbal communication. However, its shortcomings lie principally in its delivery as paper cards, making portability and use cumbersome. At the moment, there are a few web or mobile applications that offer image/test-to-speech capability, but the user interface is either too overwhelming, expensive, or too reliant on text.[1]

We believe there is a need in the market for able-speak: a simple, portable, and adaptable that is both intuitive and empowering to use.

III. SPECIFICATIONS AND DESIGN

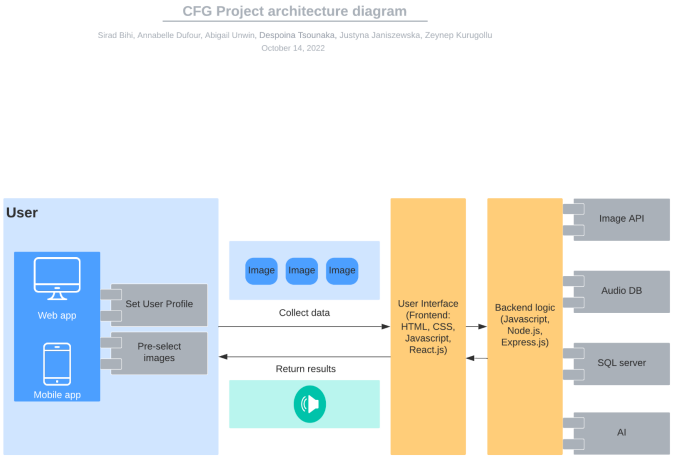
- **Requirements (technical and non-technical)**

To build our web application, we used the React Javascript library to facilitate the application's structure. While there are multiple pages in the application, the composition of each page is very similar, and so React components and their ability to be replicated would prove extremely useful in efficiently scaling the application. The application would also rely on a text-to-speech API.

[1] https://apps.apple.com/gb/app/proloquo2go/id1475643220?mt=12

- **Architecture**

The diagram below describes the original plans for this application. Time limitations did not permit us to realise several features, such as a login page. The login page was intended as a means to tailor the subsequent pages to the particular profile of the user, using saved settings to personalise the experience. Instead, we have restricted ourselves to demonstrating the composition of short sentences with no more than two inputs. And while we derive the contents from needs and activities in a home setting, this app could be easily extended to other settings, such as a classroom or a voyage.



IV. DESIGN

Our design approach was informed by the dynamic profile of our intended user: they may be young or old, with cognitive or physical disability, or perhaps even someone who does not speak English and requires translation. To meet all of these needs, we needed our app to be simple and intuitive, relying as little as possible on text only, and unencumbered by too much information. Design could not interfere with function.

- **Colour**


The dominant colours we chose, Pacific Blue (#19AECB) and Lemon Curry (#CB8F19), were selected based on colour theory, colour psychology, and creating a better user experience. Pacific Blue inspires trust, reliability and security, enticing the user, while its split-complementary colour, Lemon Curry, was chosen for its calm and bright nature, which inspires positivity and success. Rose Red is another split-complementary colour used in text. Split-complementary colours were chosen since they enhance the simple and to-the-point nature of the product and are pleasant to use.

- **Font**

The logo is set in *Brandon Grotesque* (medium weight) for its functionality and legibility, which suits the purpose of the product. The remaining text is set in *Opens Sans*, a free sans-serif Google font. It is widely considered one of the best fonts to use both in small and large sizes due to its amazing legibility and simplicity. The font is highly recommended for web and mobile use and is highly accessible.

- **Layout**

We chose the *F-pattern* as our UI design layout to ensure a better user experience. The design is made to complement and promote simplicity with minimised use of text and picture/icon and button prioritisation.

- **Heuristics**

Our project's design can be outlined by addressing the 10 usability principles in Nielsen and Molich's heuristics:

1. **Visibility of system status:** Our design prompts the user to select an image from a table of images populating the bottom half of the page, then drag it to the top half of the page to form a sequence of image buttons. This click-and-drag functionality provides immediate feedback to the user's activity. They will hear an audio description of the card, and see what has been selected, allowing for the identification of errors and self-correction, before finalising the image-to-speech function using the microphone button. By limiting the choice of images, the length of the image sequence, and functions, we make the process as simple and rapid as possible.

2. **Match between system and the real world:** Our app assumes the user is already familiar with manual non-verbal communication tools, such as the PECS (Picture Exchange Communication System)[2], or some other form of communicating their needs, except that our application offers more clarity and speed to the process.

3. **User control and freedom:** We restrict the number of images available to prevent overwhelming the user. As a possible extension in the future, we could introduce further personalization through the setup of a user profile, the ability to add tailored images, and utilise artificial intelligence to save previously used expressions.

4. **Consistency and standards:** We commit to a uniform and intuitive presentation across the application that does not distract from its purpose.

5. **Error prevention:** The click-and-drag process allows the user to consider the intended message, and identify any errors, at least twice before translating to speech: first, while selecting the individual images, and finally once the selected images have been dragged to form a sequence. Also, every card has an audio output when selected, allowing the user to verify its audio output before dragging.

[2] https://nationalautismresources.com/the-picture-exchange-communication-system-pecs/

6. **Recognition rather than recall:** Our application design is consistent with current popular style presentations, having a clean and simplified interface. Images are unambiguous and typically accompanied with text for greater clarification. Moreover, this application is designed as an improvement on manual communication tools that the user should already know.

7. **Flexibility and efficiency of use:** The simplicity of the application and the immediate feedback loop (select images translated to speech) recommends itself to both the experience and inexperienced user, as both can self-correct and continue to modify the image sequence until they find the desired speech output.

8. **Aesthetic and minimalist design:** As we do not want to distract from the critical nature of this application's intended use, we keep design and embellishment to a minimum.

9. **Help users recognize, diagnose and recover from errors:** While we minimise any possibility of error, we do not yet have clear plans for when the user does raise an error message.

10. **Help and documentation:** Ideally, the application would feature a "help section" with a video tutorial on how to use the service. And, conforming to our image-based approach, the help and documentation information would be presented in images or videos as much as possible instead of text.

V. IMPLEMENTATION AND EXECUTION

- **Development approach and team member roles**

Once we agreed on the project objective, we decided to categorise our operational requirements according to UX/UI and frontend, API and backend, and documentation. As we originally began with 6 members, we agreed to pair up and focus on one of the three categories: Annabelle and Despoina to work on UX/UI and frontend development, Abigail and Justyna on Javascript development, and Sirad and Zeynep on API and backend research.

As the project progressed and following Annabelle's withdrawal, Despoina and Abigail worked in parallel to develop the frontend, UX/UI design and React components. Sirad integrated the text-to-speech API, and Justyna composed the GitHub readme.file, which was refined by Despoina. Finally, Justyna and Zeynep conducted testing modules. Every member contributed to the documentation, and Sirad did the final editing.

- **Tools and libraries**

We decided early on to build the website with the *React library* and use *VSCode* as a tool. As the structure of the pages throughout our application is mostly uniform, we knew that the use of React components would be extremely beneficial, as a MainCard.js file could host the

main structure and the rest of the pages would use this component, and with the use of props they would be built accordingly. We also used the *React router library* (react-router-dom) in order to create routes for the pages. For the building of our MainCard.js component, we used *React Bootstrap,* which was further used in the structure of the individual pages. Lots of CSS was applied on top of everything.

Additionally, we used *content delivery network (CDN)*, which gave us the ability to use an image link from icons8.com. This was used to fetch the desired icon for our cards, so there was no need to store images.

Early research indicated we could call a text-to-speech API either through the backend or frontend. Ultimately, we decided to use the *Web Speech API*[3] for its simple implementation through the frontend. Resorting to the backend in this case would require unnecessary complexity to the code and make it more vulnerable to error.

Similarly, we decided against the Redux library as the minimal scale and complexity of our application did not merit its use. Redux would unnecessarily bulk up our code and make our application more vulnerable to error.

Finally, we used *chrome developer tools* to inspect our application which helped a lot to understand how everything was displayed. This helped us with our attempt to make our application as responsive to other devices as possible. *Jest* was used for testing purposes.

- **Implementation process (achievements, challenges, decision to change something)**

The main attribute of our application is the card, which thanks to React we have made this one card reusable and managed to populate multiple cards to individual pages. By using React Bootstrap, not only helped to create the main card, but in the individual pages we managed to display a set of cards in grid through the use of a <Container> that would consist of Rows and Columns.This was enhanced by extra CSS rules. Implementing the correct display without React Bootstrap was a challenge in the beginning.

We agreed that our application is purely a communication tool and needs to be simple. Thus, we decided not to apply a navigation bar as we need the interface to be as clear as possible for the user. We had to assume that the user is somewhat familiar with technology as well.

This project was a challenge for all the team members as we all had to research and study a lot on our own time in order to achieve the desired results.

- **Agile development (did the team use any agile elements like an iterative approach, refactoring, code reviews)**

Throughout this project we have been using Slack to communicate with each other, share ideas, thoughts and suggestions. Before we actually started building the application, we made sure that we were clear on what we were aiming our application to be. We collectively

[3] https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API.

decided what would be the ideal application, but agreed to start building small, and only if the time permitted to enhance it even more.

We followed agile methods such to break up the project in small parts and delegate tasks to each other. As a team, we also shared our strengths and weaknesses early on, just so we can have an overview of who can do what, etc. We set up Trello as well in order to try and keep track of our work there, but eventually didn't use it as we found that our regular team meetings were more effective and direct. Our meetings were taking place via Google meet and we would share our progress there.

Additionally, we've been using Google docs for project homework assignments and the final documentation. This way, each member could contribute and collaborate with each other. For version control we used GitHub. Each member would push their code into their own branch and only after communicating with each other, we would then merge the changes into the master branch.

We have been refactoring and reviewing the code constantly along the way. We also made sure to use accurate names to our components as well as our props to make our code readable. We have been reviewing the code via pull requests on GitHub too.

Our sprints:

- 1st week: Brainstorming, settling to our idea, first wireframes, choosing colours/fonts, image research, finding source for icons
- 2nd week: Finalising colours, designing mockups and prototype
- 3rd week: Setting up GitHub, setting up React App, React reading/experimenting
- 4th week: Finalising code, testing, completing the documentation of our application.

- **Implementation challenges**

    - Understanding the scope of the project

The application was originally intended to be multilingual, but that was quickly abandoned as we identified it as an additional feature, and not necessary. Able-speak is presented as an English-only format; however, now that the structure of the application is established, a multilingual function could be added relatively easily.

To further limit the scope of the project, we pre-set the environment as a home setting, and the activity categories to "Food and drinks", "Activities," "Needs," and "Emotions." Furthermore, for the purposes of demonstrating the image-to-speech function, we have limited the number of cards that could compose the sentence to only two.

    - Understanding and deciding what would be important for the user

We assumed the user would utilise the application under some time-pressure, and so the user interface is minimalist and direct. A future revision could introduce the use of artificial intelligence, or a means of storing the most frequently-used expressions, in order to expedite the process even more.

- Designing functional interface

Our team approach was highly collaborative. We would discuss and decide on aspects, such as the user interface, during our regular group calls, and the designated member(s) would work independently to realise our ideas.

- Coding/resolving issues

Resolving coding issues were similarly addressed in a collaborative approach. For example, Abigail and Sirad worked together in composing code for the complete translation of the image sentence.

- Testing

Until the final hours, testing proved to be very difficult. While Zeynep was able to conduct a thorough manual testing module, JEST was almost beyond the ability of our team members, given the short time frame and self-study required.

VI. TESTING AND EVALUATION

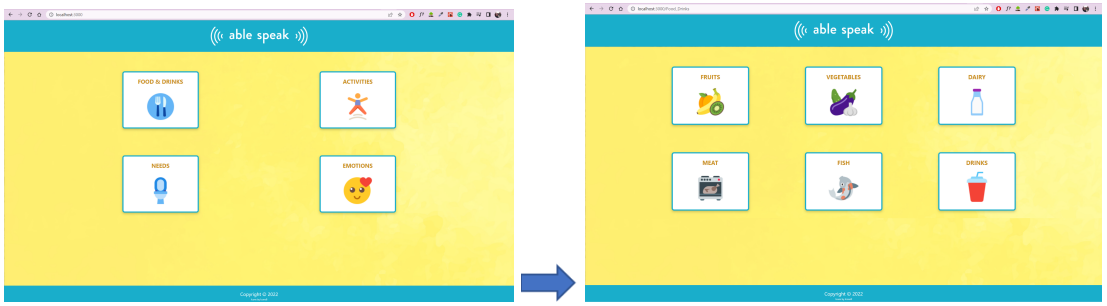
■ Testing strategy

Our testing strategy for this project is manual testing, however we also plan to test our app using JEST if time permits. The resources for manually testing the able-speak app includes a working laptop, Visual Studio Code for starting the app and a web browser.

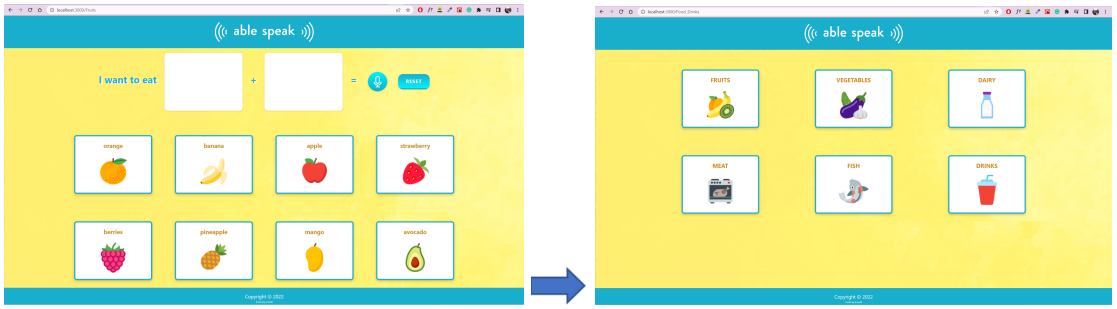
The list of features to be tested includes:

- The smooth running of the able-speak app and the transition from one web page to the next when the user selects the four options on the main webpage
- The correct speech function when the user selects a card on the main webpage and the other following pages
- The user is able to go back to the main webpage once they have finished constructing their words in the following webpages (checking the functionality of the back button)
- The user is able to drag and drop cards to construct sentences
- The user is able to click the microphone button for the speech function and the correct speech is outputted for the user
- The user is able to click the reset button to remove the selected cards for constructing other cards
- The user is able to construct a sentence with one card
- The user is able to construct a sentence with both cards

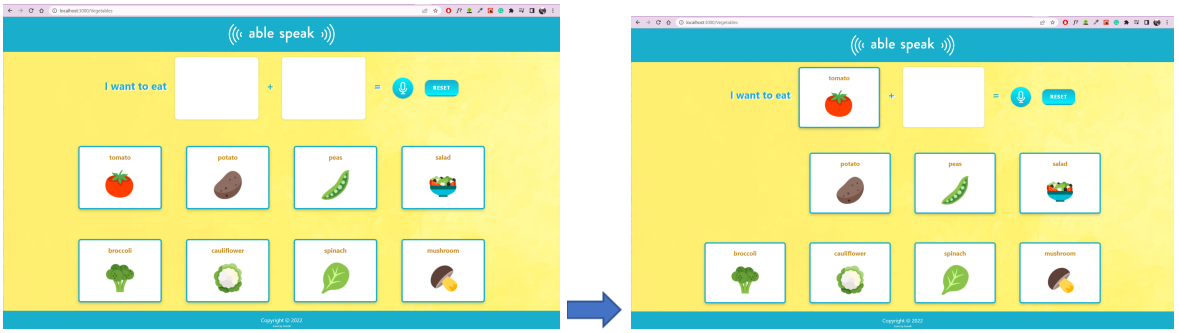
■ Functional and User Testing (Manual Testing)



When the tester selected the food and drinks card, they were able to successfully shift to the next webpage as shown above. By using the hover on functionality and having a visual cue for selection, the tester was able to easily identify their choice. Their selection was further confirmed by the correct audio cue. For example, when the tester selected the “food and drinks” card, the audio read out “food and drinks”. This was the same with all cards on the main webpage and the other following webpages.

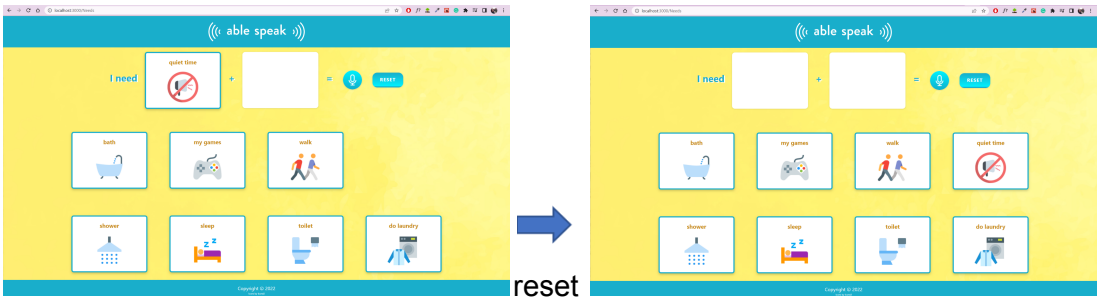


The tester has selected the back button on the top-left corner of the screen to ensure that users could shift to previous pages and navigate to other webpages. It can be confirmed that the back button is functional within all pages of the app.



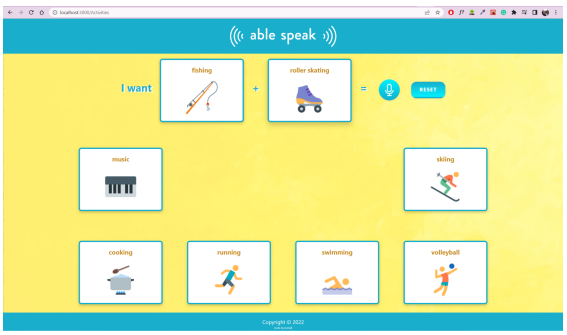
The tester was able to successfully drag and drop the selected cards into the constructor without any issues. It can be confirmed that all cards in every different category were able to be selected, dragged and dropped into the constructor with ease.

The tester selected the microphone button in the constructor to verify that the text-to-audio is working to standards. It can be confirmed that the speech function is working as the tester was able to clearly hear the constructed sentence e.g. “I need quiet time”. The speech function has been checked for all cards in each different subcategory and it can be confirmed that the tester has heard the correct response.



The tester has selected the reset button to verify that the selected cards are automatically removed from the constructor for it to be reset. It can be confirmed that once the reset button is clicked, the previously selected cards are removed from the constructor sentence, and it is left empty for new cards to be dragged and dropped into the constructor. This has been tried and tested for all other selected cards in each different subcategory within the able-speak app.

The tester has selected only one card to construct a sentence to determine whether the text-to –audio will provide a response. The tester can confirm that it is possible to choose only one card to construct a sentence and the text-to-speech function is able to provide a response, for example in the picture above, the tester can verify that “I want to eat mango” could be heard. This has been tried and tested for all other selected cards in each different subcategory within the able-speak app.



The tester has selected two cards to construct a sentence to determine whether the text-to-audio will provide a response. The tester can confirm that the text-to-speech function can provide a response after both cards are dragged and dropped into the constructor sentence. It could be verified that the correct text-to-speech audio of “I want fishing and roller

skating” could be heard as shown in the picture above. A slight improvement could be made in the “Activities” category to construct a sentence with the correct grammar. Instead of “I want fishing and roller skating” it would be more convenient to construct the sentence as “I want to go fishing and roller skating”. This could be resolved by assigning “to go fishing” to the “fishing” card. Similarly this could be applied to other cards, for instance instead of just “running” it might be more useful to assign “to go running” to the “running” card.

Finally, the tester is able to return to the main page at any point by clicking on the logo.

- **JEST Testing**

Automated tests for able-speak app were run using JEST and React Testing Library. In the short amount of time that we had we did a great amount of research into using JEST to complete our project. Our biggest hurdles were writing the correct syntax and adding data-testid attributes for each component to identify DOM for testing purposes. After many attempts we successfully tested chosen Sentence Container and MainCard components of the app.

The following were tested:

- Correct number of MainCard components rendered in the home page (App.js),
- Correctly rendered MainCard component, refresh button and microphone button,
- The functionality of the refresh button

If time had permitted, future work on the unit tests could have been implemented. This includes more extensive unit test coverage; whilst being able to fully test the rendering and functionality of the refresh button, we were not able to fully test for every functionality of the other components. More complex features such as the speech synthesis utterance of the MainCard component would need to be tested.

- **System limitations**

Some system limitations are present by design; for example, the limited environments and categories (ie. vocabulary). Other limitations are a result of time pressures (e.g. an “About” page, a “Help” section, and a “Login” setting). Furthermore, one team member withdrew unexpectedly and without communication, forcing other members to assume her responsibilities.

- **Extensions**

Due to time restrictions and limited experience, we were unable to incorporate the following features, which we believe would greatly enhance our web application:

  - Option to switch between positive and negative statements (e.g. “I want a banana” versus “I don’t want a banana.”)
  - Less reliability on text: in some instances, we have text-only functions (e.g. the “Reset” button). Using more images to signify meaning would make our application more universal; rather than translating the text on the user interface, we could just use a different language Web Speech API.

- A login page: to allow for personalisation, such as the use of personalised images (e.g. “memojis”) and environments or vocabulary.
- A “Help” section: to demonstrate the use of our application through a video tutorial

VII. CONCLUSION

We all agree that this project experience has been both extremely challenging and gratifying. We have been able to implement all of our learning throughout this course, but most importantly what we learned through self-study, especially in greater knowledge of Javascript, React, JEST and the use of APIs.

Nevertheless, we have created an application that is a genuinely viable product. It addresses a real need of a significant minority in our communities, and can easily scale to the circumstances of the greater community (e.g. using image-to-speech capability for real-time language translation.) While there are some similar applications presently on the market, able-speak distinguishes itself from the competition for its simplicity, intuitiveness, and portability.