Blacksburg, VA 24061
CS 4624 – Multimedia, Hypertext, and Information Access

FreeSpeech4VT

Instructor: Dr. Edward A. Fox

Client: Matthew Newton

Authors:
Andy Cho
Chaitanya Gupta
Samuel Oh

May 8, 2022

Table of Contents

Table of Figures **4**

1. Abstract **5**

2. Introduction **6**

2.1 Background 6

2.2 Objective 6

3. Requirements **7**

3.1 Minimum Requirements 7

3.2 Supplementary Features 7

4. Design **8**

4.1 Existing Application and Wireframe 8

4.2 Design of FreeSpeech4VT 9

5. Implementation **10**

5.1 iPadOS Application 10

6. Users' Manual **11**

6.1 iPadOS Application 11

6.2 Tiles Page 12

6.3 Settings Page 13

6.3.1 Add Tile 14

6.3.2 Edit Tile 18

6.4 References Page 19

7. Developer's Manual **20**

7.1 iPadOS Application 20

7.2 Environment 20

7.4 Implementation Details 20

8. Lessons Learned **26**

9. Future Work **27**

10. Acknowledgements **28**

11. References **29**

Table of Figures

Figure 1: Existing Application 7
Figure 2: Wireframe of Home Page 8
Figure 3: Microphone Permissions at First Launch 11
Figure 4: Tiles Page 12
Figure 5: Settings Page 13
Figure 6: Add Tile Options 14
Figure 7: Add Tile Options Populated 15
Figure 8: Additional Tile Options 16
Figure 9: Tile Addition Success 16
Figure 10: New Tile Page 17
Figure 11: Edit Tile Page 18
Figure 12: Edit Tile List 19
Figure 13: Edit Tile View 19
Figure 14: Delete Tiles Alert 20
Figure 15: References Page 21
Figure 16: Git Repository 22
Figure 17: ContentView.swift 25
Figure 18: SpeechGrid.swift 26
Figure 19: AddTile.swift 27
Figure 20: EditTile.swift 28

1. Abstract

Over the course of his career, Mr. Matthew Newton, the coordinator of Assistive and Education Technology at Virginia Tech, has been working with applications and assistive technology in order to aid those who require non-verbal communication, to receive the means to do so. The FreeSpeech4VT project was launched under the direction of Mr. Newton to provide a free-of-cost tablet application that would allow users to easily communicate using type-to-speech functionality. Our goal as a team was to create an application that could provide basic and advanced implementations of features that paid-applications offer, while also promoting user customization of the application itself.

The project consists of one mobile application that pulls from the device's local data and is able to store data into the device's memory based on user input of words/tiles. Although cross-platform applications could provide more flexibility and accessibility for users depending on what operating system their devices run on, our team found it best to implement a thorough iPadOS application due to the high frequency of iPad usage for communicative purposes.

This application is designed for anyone who may struggle to communicate verbally, both temporarily or more long-term. There were some design choices to be more friendly towards those who may have difficulty with motor function as well. The range of people that can use this application is immense; it can be anywhere from someone who is unable to speak at all and may have some motor function issues to someone who has laryngitis and is able to type out sentences just not speak or speak loudly for some time.

We began by meeting with Mr. Newton to discuss his vision. Because ergonomics with the target user-group in mind was so important, we spent a lot of time on iterative design and wireframing, getting feedback and making improvements, and finally getting approval. We then began implementing the design as well as basic Text-To-Speech (TTS) functionality. After that, we implemented more customizability-centric functionality to allow both ease-of-setup for caretakers and ease-of-use for users. We obtained feedback via bi-weekly client meetings.

We have delivered an iPadOS application, this report, a final slide presentation, and a video walkthrough of the application in use.

2. Introduction

2.1 Background

The Americans with Disabilities Act defines a disability as "...a person who has a physical or mental impairment that substantially limits one or more major life activities…". [1] Some disabilities include difficulty with mobility, cognition, independent living, hearing, vision, and even self–care. [2] Speech and communication disabilities are among these; according to the National Institute of Deafness and Other Communication Disorders in 2016, nearly 8% of children of age 3-17 in the U.S. had a disorder related to voice, speech, or swallowing. [3] Considering the importance of language exposure to a child's development, these disabilities pose a serious threat to these children. Allowing those with speech impairments to practice language is not only critical to their everyday wellbeing but also critical to their development.

2.2 Objective

While there are some solutions and alternatives to assisting non-verbal communication, many necessary features such as text-to-speech and user customization are locked behind premium versions. With the direction of our client Mr. Newton, our primary objective was to create an application that provided quality text-to-speech functionality and user customization for free. This objective was supplemented by specific features that Mr. Newton requested from personal experiences with existing applications. Our primary target audience were children with communication disabilities, but we remained aware of accessibility features that could benefit (or hinder) older users.

3. Requirements

In terms of requirements, we were given a loose structure of what was expected and so conversed with Mr. Newton to specify the exact requirements of our deliverables. After consideration and design discussion, we narrowed down the minimum requirements and supplementary features to what is below.

3.1 Minimum Requirements

The minimum requirements are as listed:

- Text-To-Speech functionality with tiles
- Tile grid with each tile containing a word and an image
- Static access bar of commonly used tiles
- User customization options for
    - Tile addition
    - Tile deletion
    - Edit tiles
        - Color
        - Custom sound
        - Where it appears
            - Main grid
            - Static tile dock

Minimum requirements are what we would be responsible for in terms of solution delivery at the end of the Spring 2022 semester.

3.2 Supplementary Features

The supplementary features are as listed:

- Predictive tile display
    - When one tile is clicked, a specific subset of potentially relevant existing tiles is shown/highlighted compared to other tiles.
- Tiles split into pages
    - Reorderable
- Alternative voice options
    - Younger, older voices
    - Different accents

Supplementary features are those that either the client wanted to see implemented or things that we as a team wanted to attempt, but after some research and discussion they were either lower priority and/or didn't fit the project scope of the semester.

4. Design

4.1 Existing Application and Wireframe

We first began our design process by studying the existing application given to us as a reference. [4] The design consisted of multiple tiles with a word and an image. Upon clicking a tile, the respective word is added to the sentence bar which can then be spoken with the action buttons at the top right. Each tile is organized in a structured grid, and may or may not be colored to signify belonging to a set (user determined).

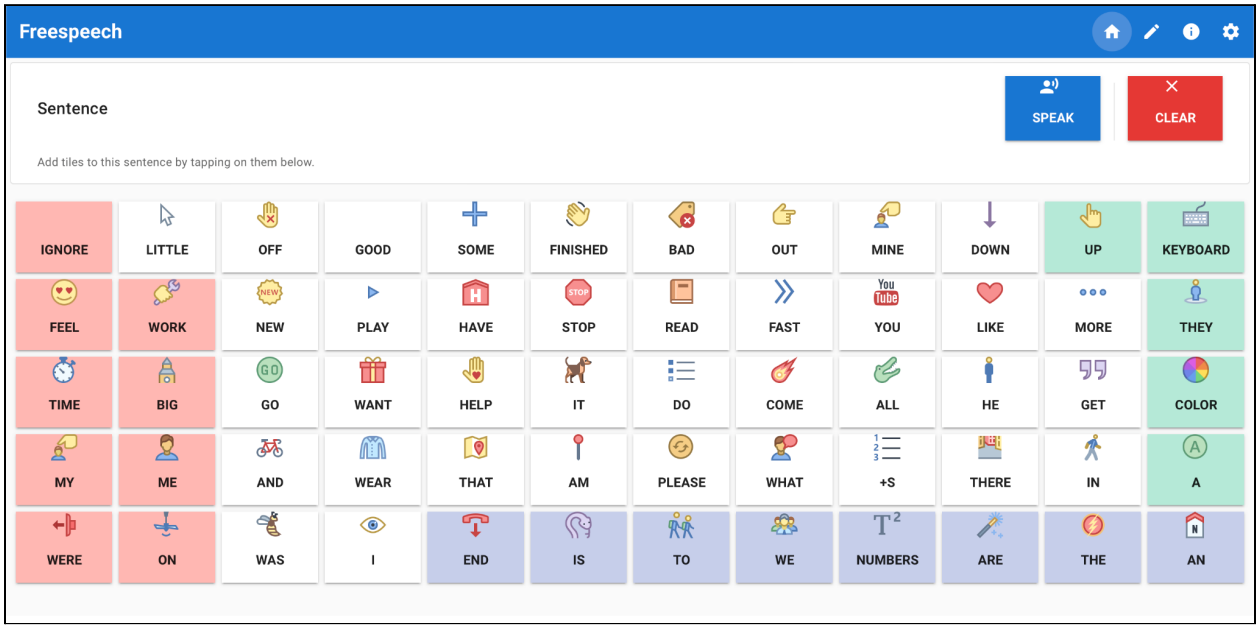

Figure 1: Freespeech Website - adapted from [4]

We drew heavily upon the existing application's design as seen in Figure 1, and transformed it to fit onto a mobile screen. At this point we had not solidified the minimum or supplementary feature set, but we still wanted to create a wireframe to present our initial ideas to Mr. Newton and receive feedback.

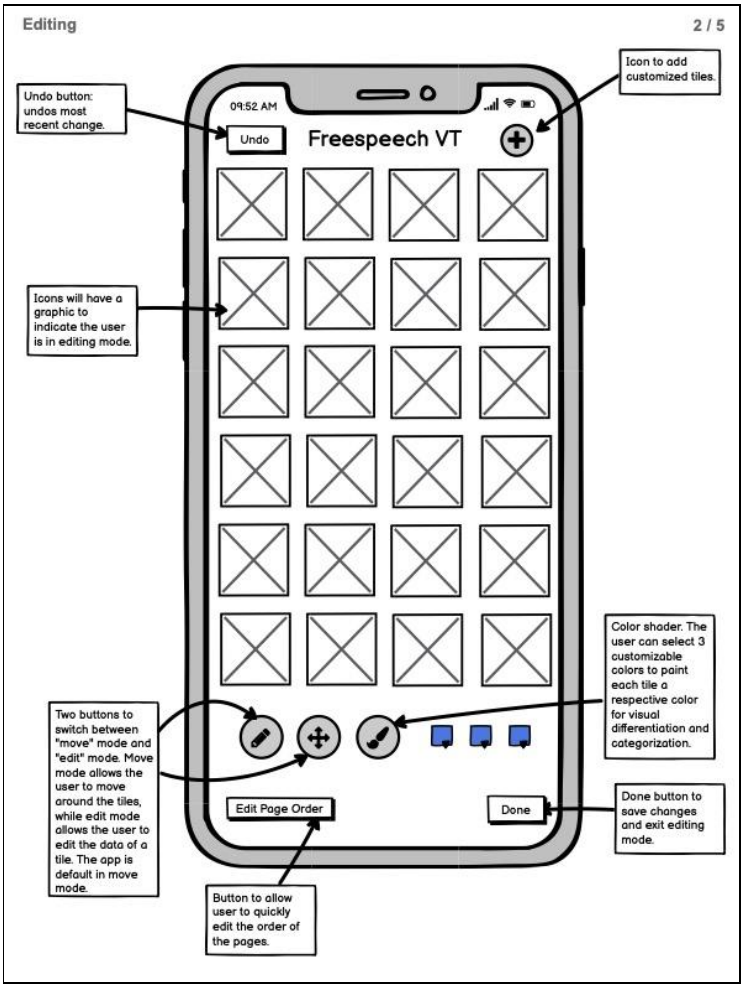

Figure 2: Wireframe of Home Page

The wireframe in Figure 2 is for the home page of our application. It maintained very similar qualities to the existing application; there was a grid of tiles that were able to be queued and then spoken using action buttons at the bottom. After presenting our initial wireframing to Mr. Newton, we received feedback on the placement of features and sizing. We also narrowed our scope to only the iPad due to relevancy.

4.2 Design of FreeSpeech4VT

The design of our iPadOS application was the biggest factor in our process, even above functionality, as the entire purpose of our application is to be ergonomic for our users. Once design principles are determined and followed, adding functionality can always be done, whereas changing design would be more difficult.

We met many times with our client during our design process to ensure an iterative design that would both satisfy the client as well as be easy to understand and use for the users and potentially their caretakers. [7] We also were able to interact with a device preloaded with several implementations from other manufacturers of what we are trying to build. But our solution had to be in a free, open-source manner rather than charging money or requiring subscriptions. Based on all the feedback and time spent interacting with existing solutions, we were able to build a general layout with some design principles we could follow and reflect throughout our application.

There is a notable lack in terms of formative evaluation (especially from a member of our target user demographic); however, we met with our client Mr. Matthew Newton on a bi-weekly basis. Since Mr. Newton works with people in our target demographic on a daily basis, we were able to ask questions, take criticism, and focus our design. Therefore, despite our lack of formal testing, his input and feedback served as relevant evaluation.

5. Implementation

5.1 iPadOS Application

The goal of FreeSpeech4VT is to create a royalty-free application that would allow the user greater access to a software system that assists in communication in the every-day life. First, we created a reference page to display acknowledgements of our client and our resources for this application. This area will display information on who data was collected from, as well as the visionary aspect of the client.

We then implemented a grid display that would first pull information of basic and frequently used words from a JSON file structure. The file contents would then be placed into core data of tile structures, which would be used to then display all the words and their given pictures onto the grid.

The grid itself was implemented through 3 different splits into the display. The first split, from top-bottom, is our sentence display. Here, the user can place the tile's word content into the sentence box, and we would use Apple's voice synthesizer in order to speak the sentence variable, where words were added into. The grid itself was implemented through the use of Swift's VGrid and HGrid structures, which line the tiles up in grids. Users can then click on each tile to trigger an event that would input the word into a sentence variable. The very bottom is for a static tile display, which allows users to scroll through a list of words that are more frequently used than others.

Finally, we implemented a settings page to place customizable options for the sake of accessibility and the freedom to choose certain details for the tile grid. In order to save user inputs into storage, we use a core data model that allows for the direct storing of tile data into device memory. This allows the user to save unique words to the device's memory so that they can be retrieved at a later time when the app is reopened. In the next couple of sections, we will delve into the features of what each section includes and looks like.

6. Users' Manual

6.1 iPadOS Application

The goal of this application is to provide a robust and customizable text-to-speech implementation to aid non-verbal disablees in communication. This application comes with initial words and pictures, and has the option to add custom words with personalized voice memos. Additionally, we included graphical notations to help users identify and group words into categories (ex., commonly used words).

After downloading the application, the application will appear on the device's home screen titled "FreeSpeechVT". At first launch, the loading page will show. Afterwards, the app will request permission to access the microphone as seen in Figure 3. This is to allow custom voice recordings for tile additions.

Figure 3: Microphone Permissions at First Launch

The initial tile data will then be loaded. Any changes made thereafter will be saved after exiting the application. This can be manually reset by uninstalling the app completely and then reinstalling.

6.2 Tiles Page

The tiles page is the default page upon application opening, and is also the main function of the application as shown in Figure 4. The page is divided into a vertically scrollable 10 x 5 grid of tiles, with each tile containing a word and an associated picture. The “Speak” and “Clear” buttons are on the left and right of the top bar respectively, with the sentence preview being in the middle. At initial launch, the sentence preview is empty. However, some input words are shown in Figure 4, for clarity.

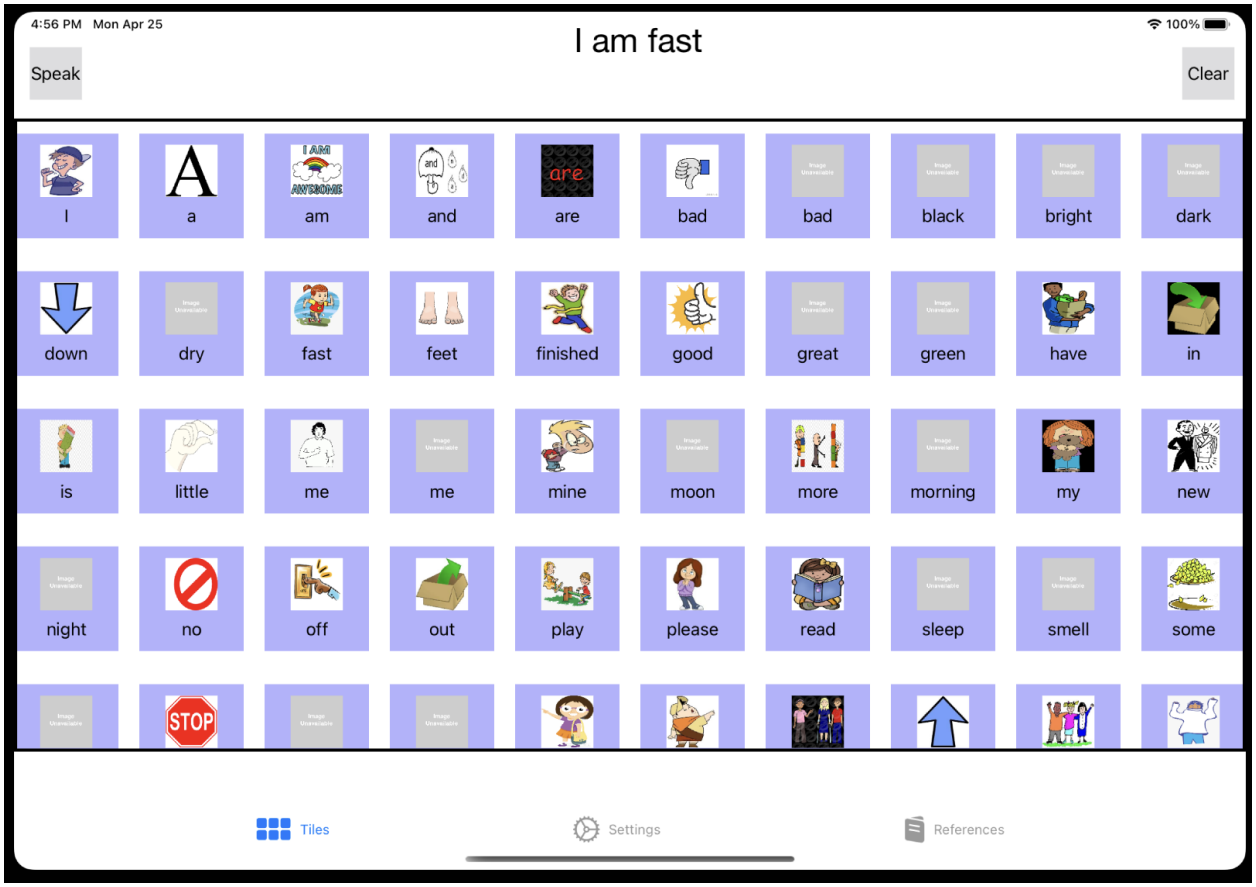

Figure 4: Tiles Page

Upon clicking on a tile, the word will be spoken and will be added to the sentence preview. The sentence will be spoken upon pressing “Speak”, and the sentence can be cleared with the “Clear” button. Additionally, the sentence preview bar is accessible by touch and words can be manually typed into the bar.

6.3 Settings Page

The second tab bar view page is the Settings page as shown in Figure 5. This page contains options for the application. These options include the add tile and edit tile functionality.

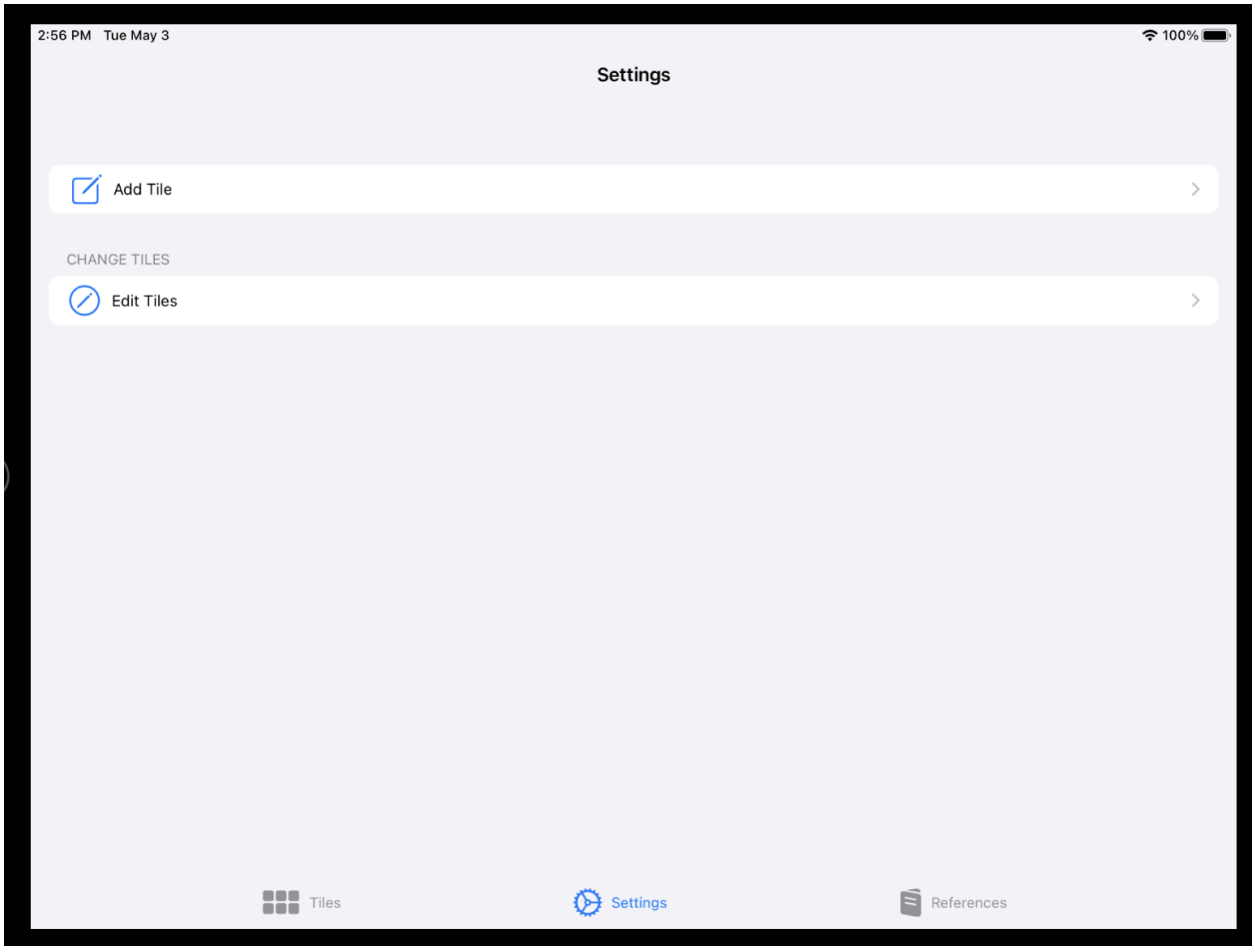

Figure 5: Settings Page

6.3.1 Add Tile

Upon clicking the “Add Tile” option in the Settings page, the view in Figure 6 is shown. This form allows the user to add a custom tile. Within the form, there are various options detailing different customizations the user can make. The user can input their word in the “WORD” text box (and clear their word with the X symbol on the right), and select the associated photo by either taking a picture with their camera or selecting a picture in their Photo Library. The associated photo is displayed in the “TILE PHOTO” section; if no picture is selected, the default “Image Unavailable” photo will be used. The user may then record a custom voice memo for the word; if this action is not performed, the word will be spoken using the default text-to-speech voice. After these fields are filled in, the values will be respectively displayed as seen in Figure 7.

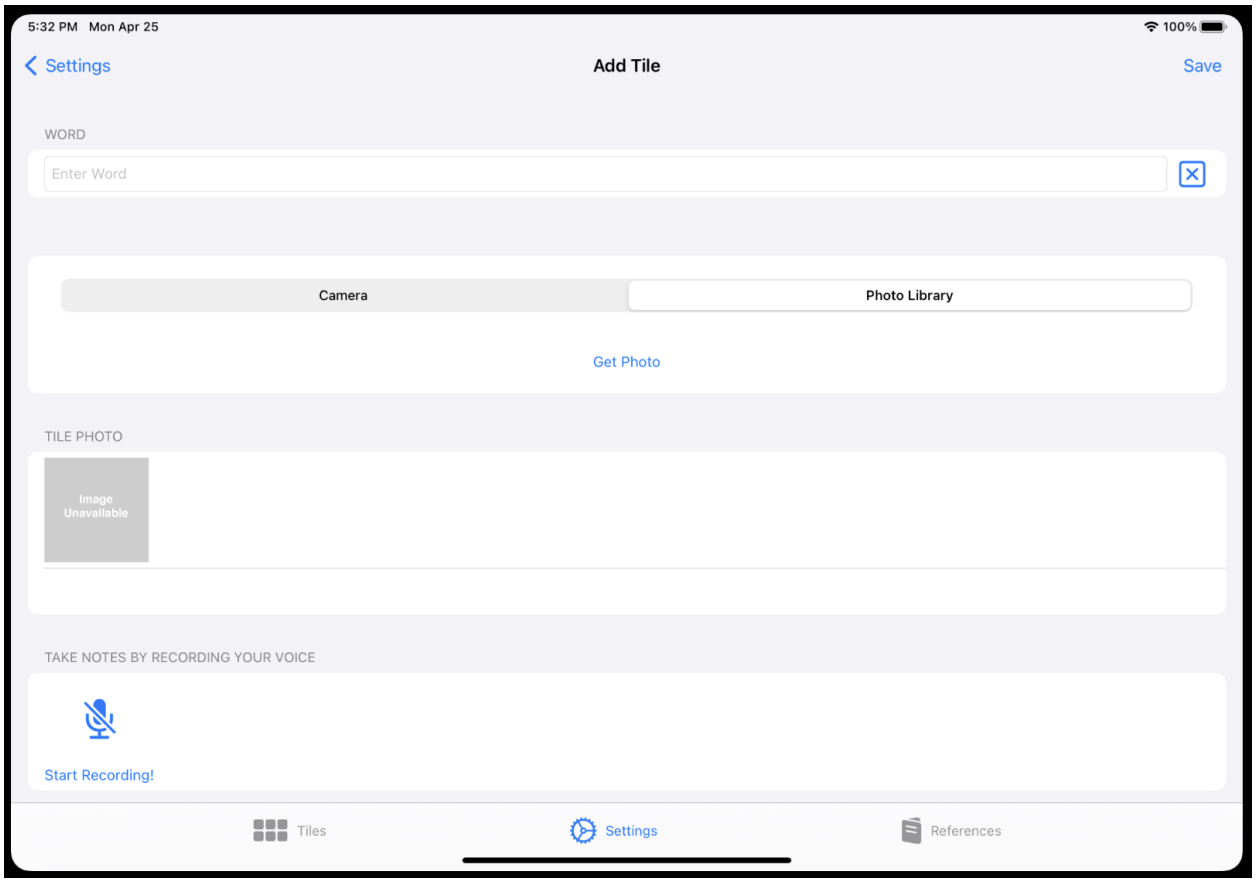

Figure 6: Add Tile Options

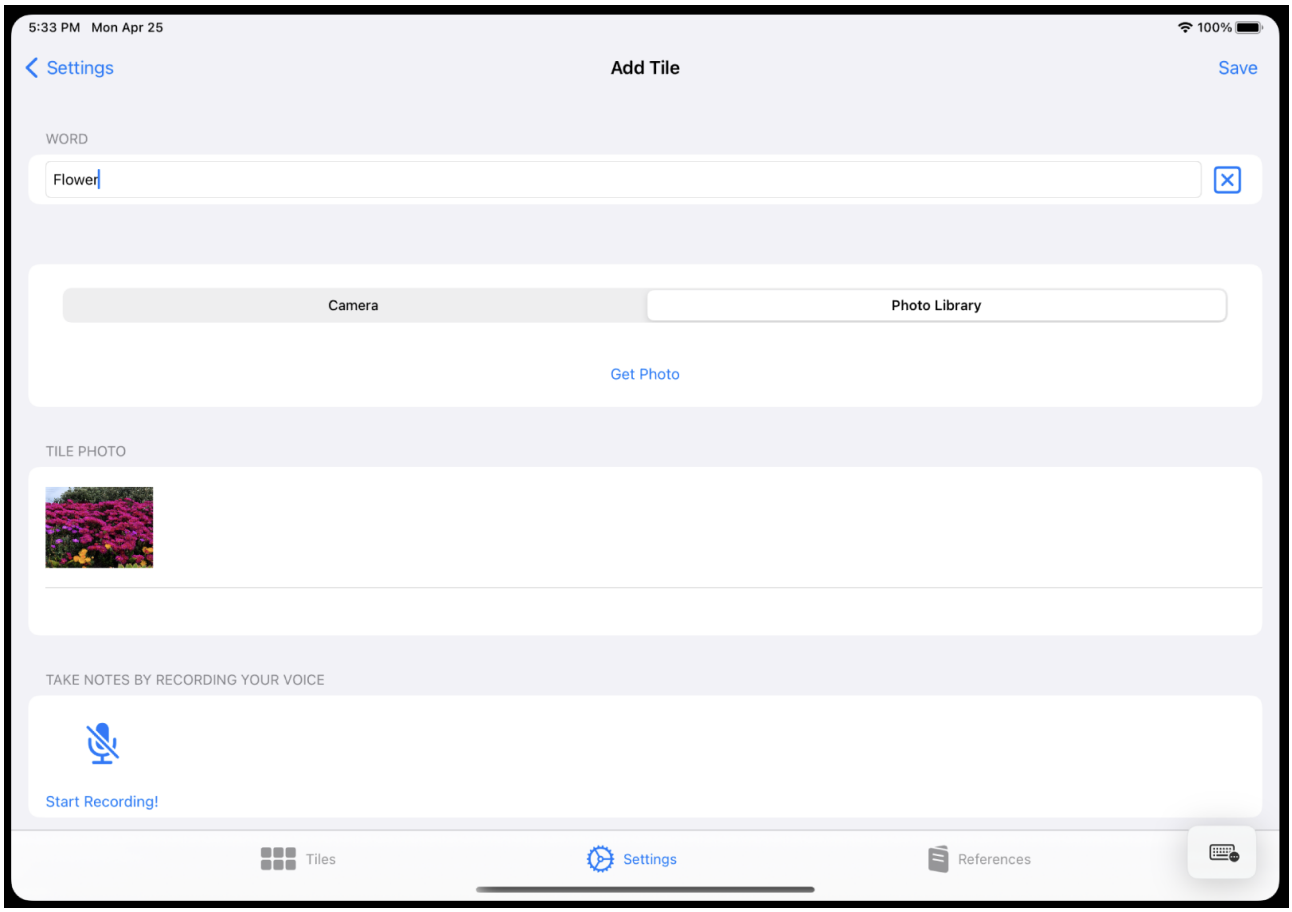

Figure 7: Add Tile Options Populated

There are additional customizations that the user can make in the "Add Tile" form as seen in Figure 8. One of these customizations is the ability to change the tile's color. A custom added tile will have a default selection of yellow, but this can of course be changed. Another customization is the "FREQUENCY" flag; selecting this option will place the tile in a custom area in the Tiles page. After the form is completed, an alert is displayed to notify a successful addition. See Figure 9.

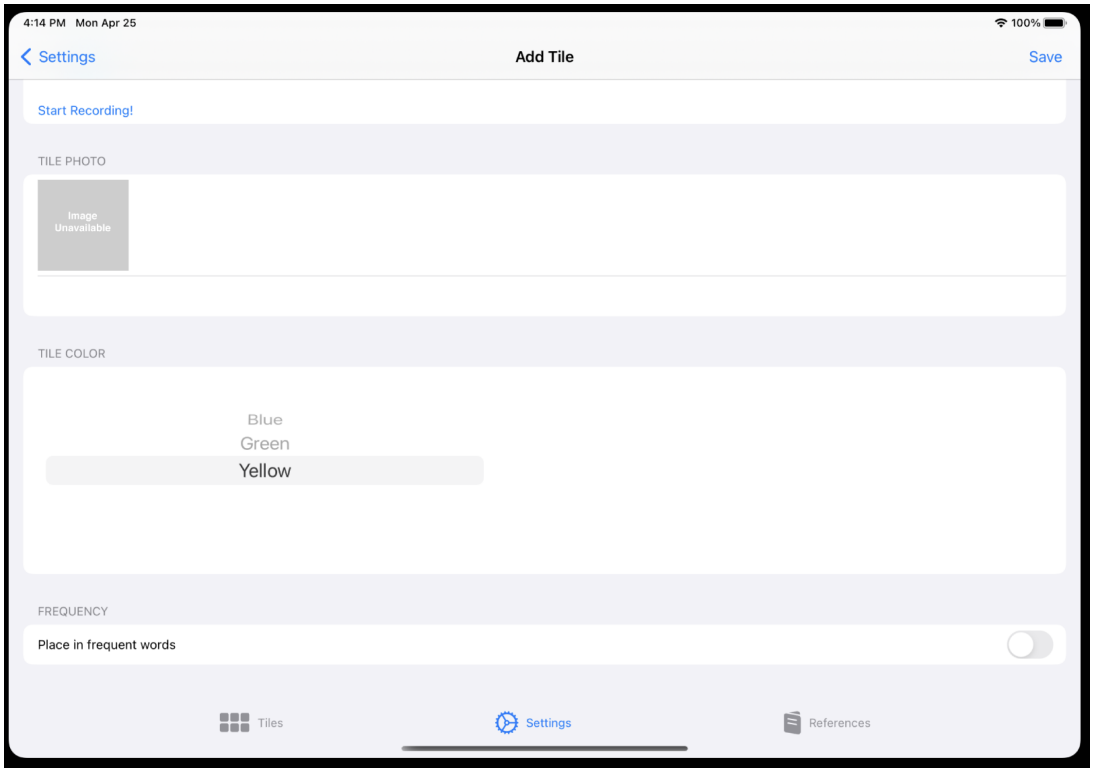

Figure 8: Additional Add Tile Options

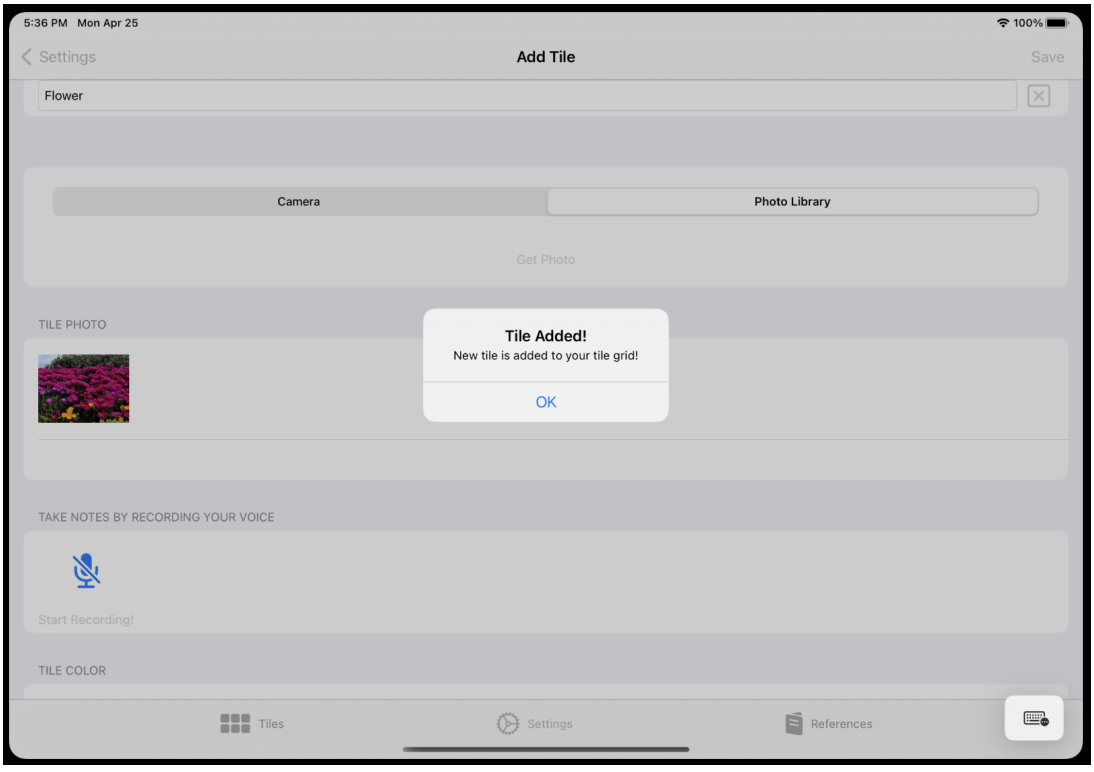

Figure 9: Tile Addition Success

After a tile is added, the Tiles page is updated with the new tiles as seen in Figure 10. The 10 x 5 grid becomes a 10 x 4 grid to accommodate for the frequently used tiles section. If there are no frequently used tiles, the grid reverts back to a 10 x 5 formation.

Note the frequently used tile section. Unlike the main tiles section, the frequently used tiles section scrolls horizontally.

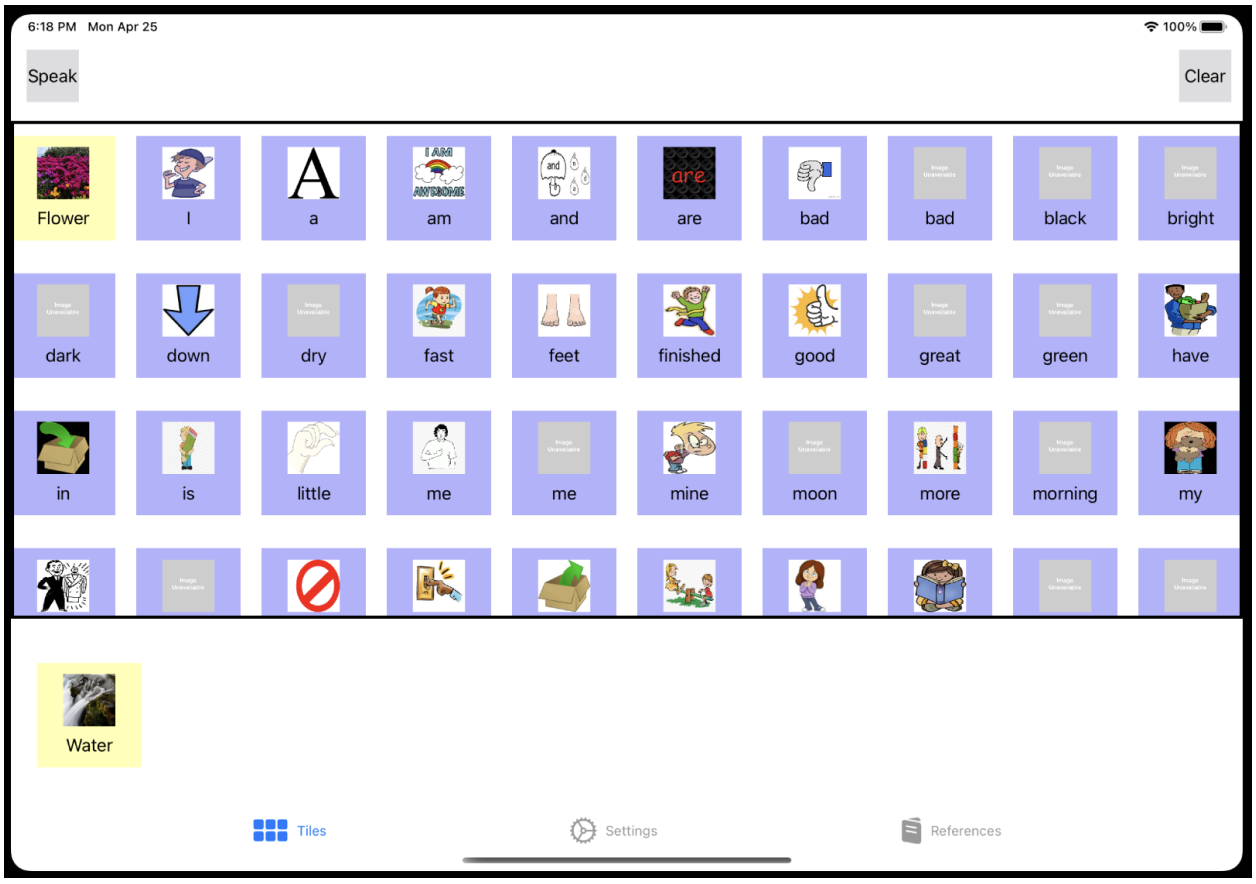

Figure 10: New Tile Page

6.3.2 Edit Tile

Upon clicking the “Edit Tiles” button on the Settings page, the view in Figure 11 appears. There are three options from this point; edit tiles in the regular speech grid, edit tiles in the word dock, or delete all existing tiles.

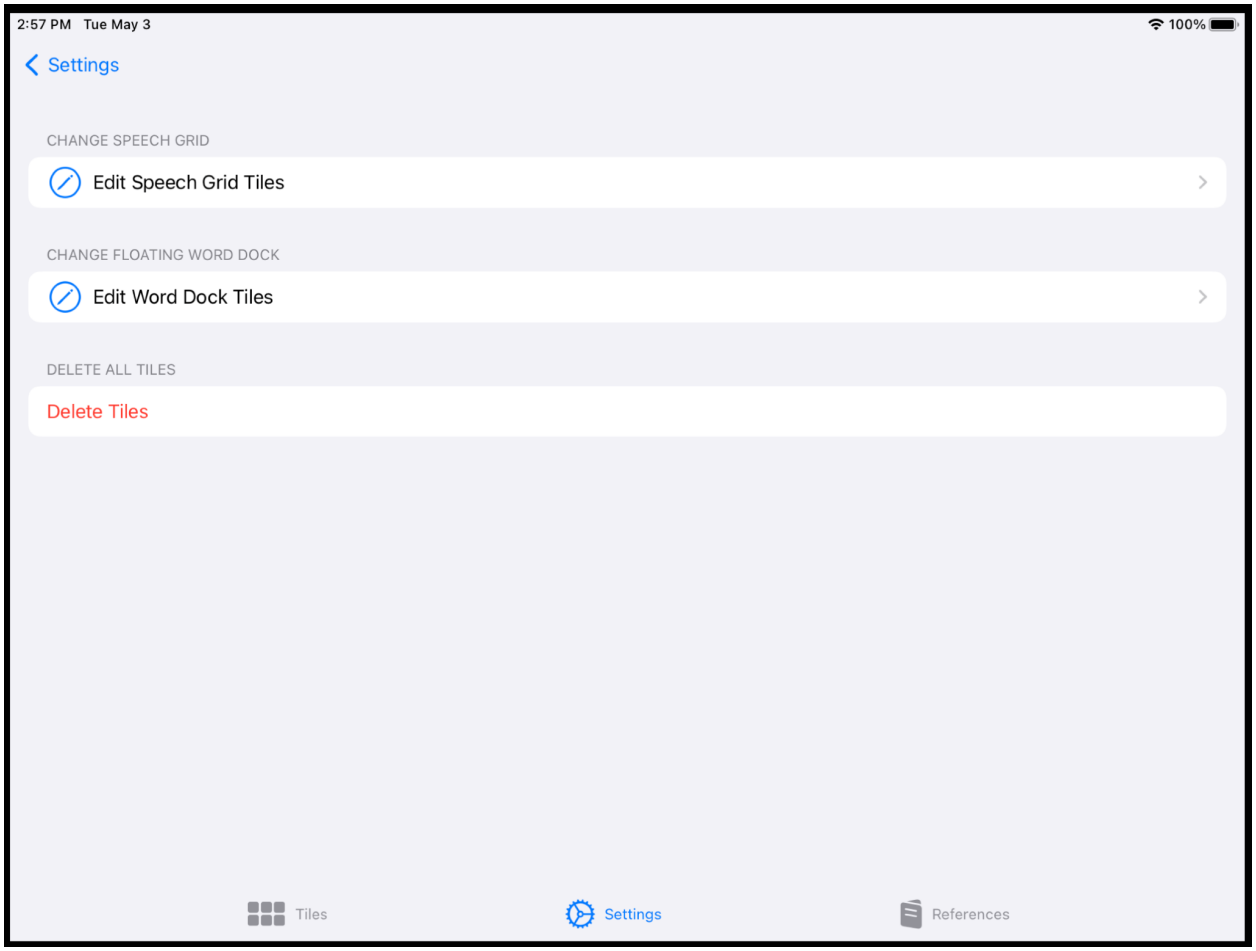

Figure 11: Edit Tile Page

Clicking the “Edit Speech Grid Tiles” will display the list of tiles in the speech grid. See Figure 12. The user can then select a tile to edit, which will display the tile’s information in an editable view as seen in Figure 13. The user can then edit the values of the tile similarly to adding a new tile and save the tile using the “Save” button on the top right. The user may also delete the tile with the “Delete” button on the top left. After either option is selected, the list of tiles of the speech grid will be returned to view.

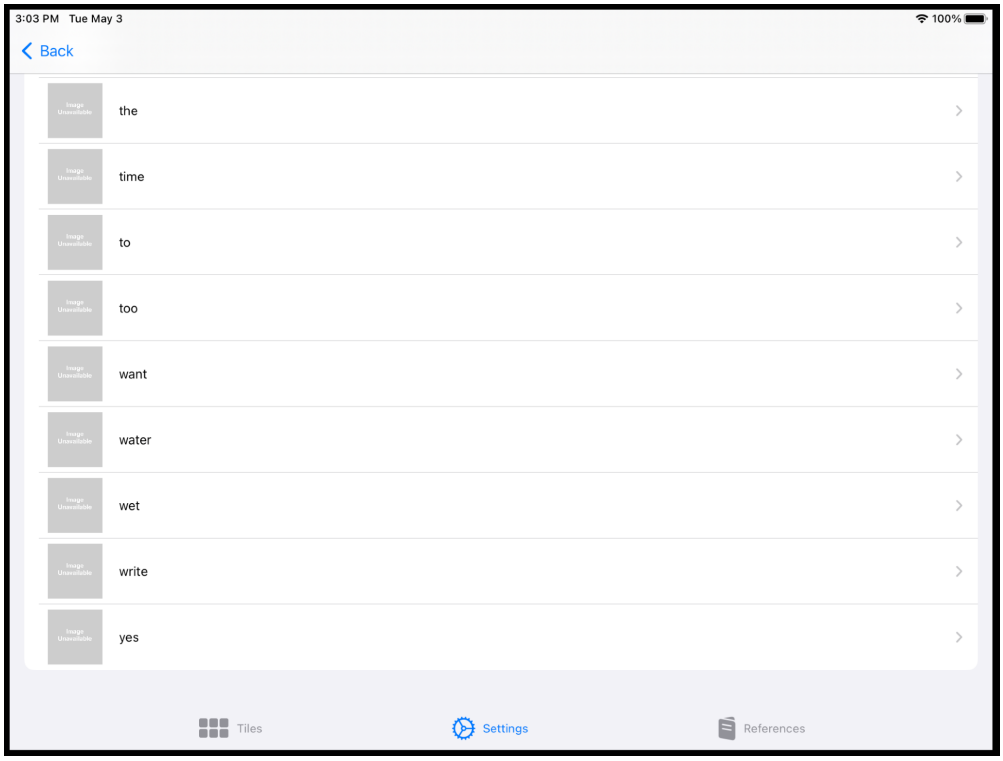

Figure 12: Edit Tile List

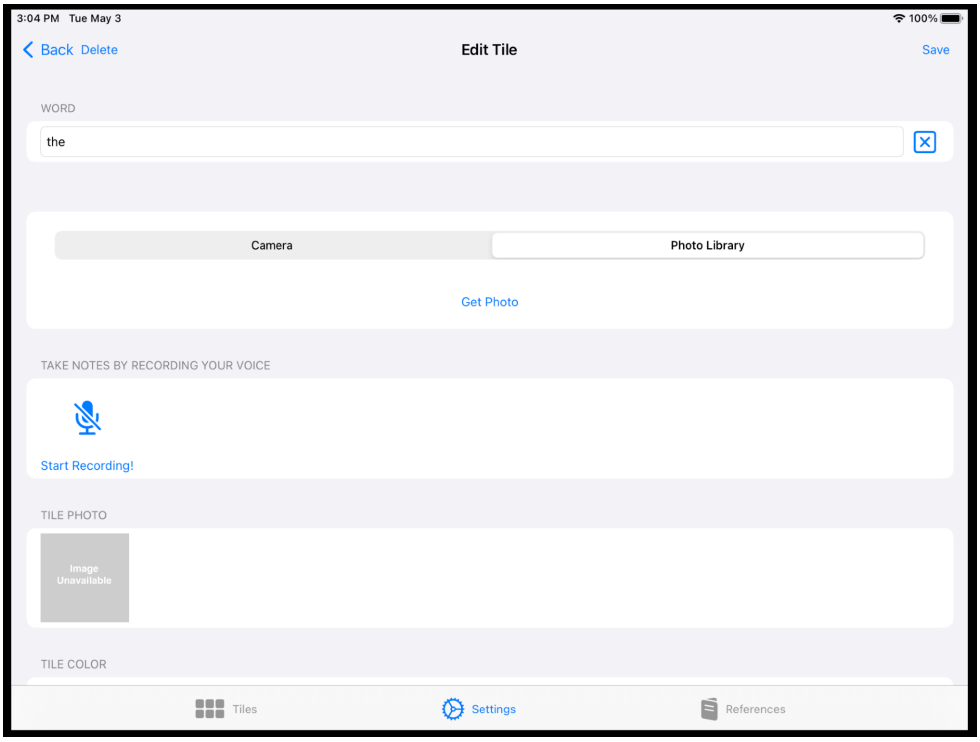

Figure 13: Edit Tile View

The “Edit Word Dock Tiles" option will bring the same views for the respective tiles. By default, there are no tiles in the word dock; therefore this list will be empty.

If the “Delete Tiles” option is pressed, an alert will appear requesting for confirmation as shown in Figure 14. If “Delete” is selected, then all tiles will be removed from the database and the user will have to manually add tiles one by one.

Figure 14: Delete Tiles Alert

6.4 References Page

The last tab bar view page is the References page. This view details a short explanation of the application and acknowledgements to our client and mentors. This page is view only. See Figure 15.

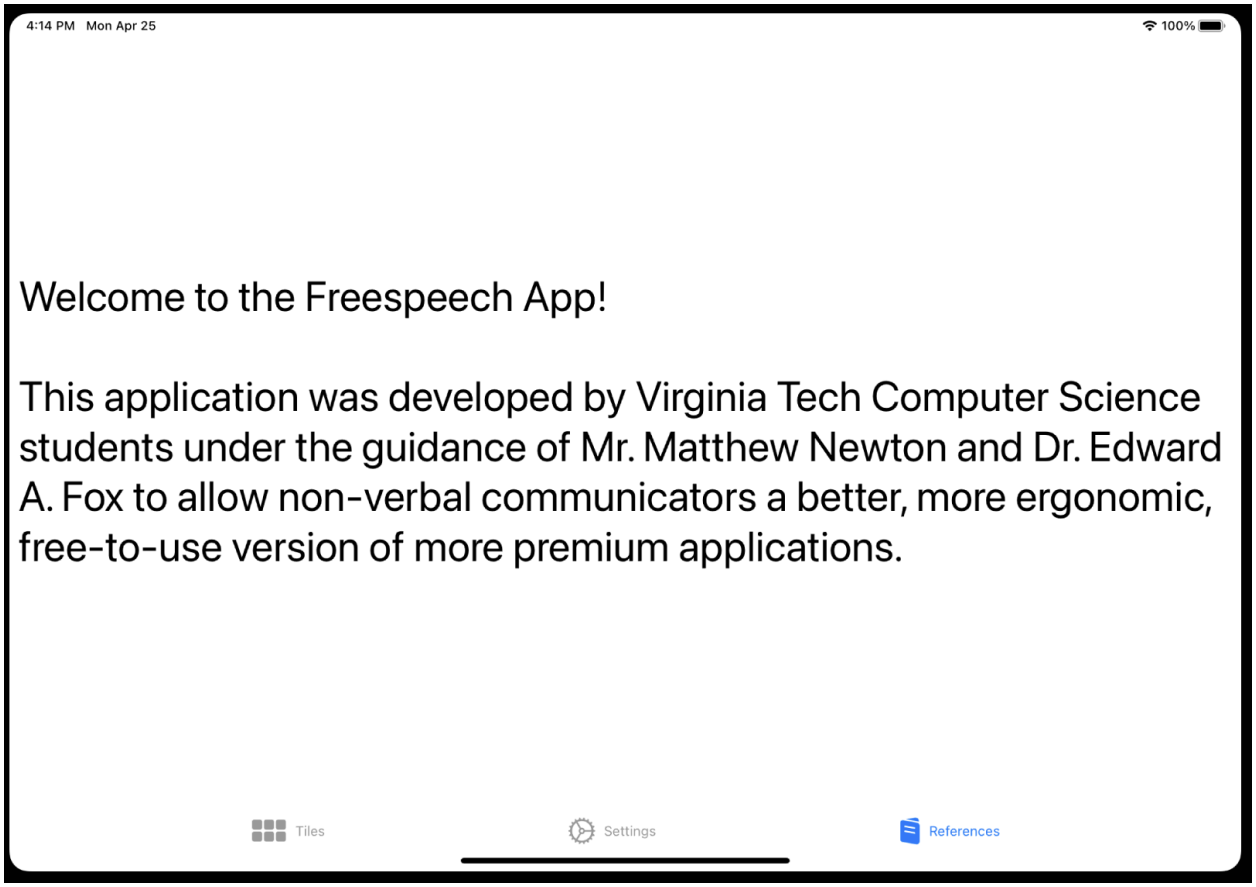

Figure 15: References Page

7. Developer's Manual

7.1 iPadOS Application

This application was developed from scratch using Xcode in the SwiftUI framework. There are elements of UIKit across the application, but most of the pages utilize SwiftUI for future-proofing.

7.2 Environment

Our application is iPadOS only, which means a developer must have access to Xcode and in consequence MacOS. Xcode is downloadable from the MacOS App Store. Our team developed and maintained the application using the latest version of MacOS Monterey (12.3.1) and Xcode (Version 13.3.1).

For this application our team utilized Git for source control. To access the source code, clone the project detailed in the Git repository. [5] The project will contain the following files shown in Figure 16.

Figure 16: Git Repository

To clone the project, click on the green "Code" button on the top right. A view will appear detailing how to clone the project. Refer to this report's references for further details on Git and Git cloning. [6]

7.4 Implementation Details

We detail the larger implementation modules of this project in this section. Please refer to the source code in the Git repository for fuller documentation, as the source code is more thoroughly documented with comments. [5]

The project contains the following files and folders (bolded), with a short description of each file:

- **FreespeechVT**
    - AppDelegate.swift
        - Performs initial launch actions
    - FreeSpeechVTApp.swift
        - Performs data saves after view switches using Persistence.swift
    - ContentView.swift
        - Creates tab view for app
    - **Take or Pick Photo**
        - ImagePicker.swift
            - Utility file to allow picture to be selected from photo library
        - PhotoCaptureView.swift
            - Utility file to allow picture to be taken from user's camera
    - **Core Data Model**
        - Persistence.swift
            - Contains function to save CoreData changes
        - Photo.swift
            - Photo object file with relevant data fields
        - Tile.swift
            - Tile object file with relevant data fields
    - **Data Models**
        - WordsData.json
            - Initial word data in JSON format
        - WordsData.swift
            - Parses JSON file and populates database with Word objects
        - WordStruct.swift
            - Word object file that holds information parsed from WordsData.json
    - **Supporting Files**
        - UtilityFunctions.swift
            - Miscellaneous helper functions (decode JSON, transform image data)
        - VoiceRecording.swift
            - Utility file to record custom voice memos
        - AudioPlayer.swift
            - Utility file to play audio (custom voice memos)
    - **Tab Bar Views**
        - **Home**
            - References.swift

        - View-only page that describes client and project vision
    - **Speech**
        - SpeechGrid.swift
            - Contains structure of tile grid and main functionality of application
    - **Settings**
        - Settings.swift
            - Tab view page to display “Add Tile” and “Edit Tile” options
        - AddTile.swift
            - Creates new tile and adds to tile grid and database
        - EditTile.swift
            - Edits existing tile and data values and updates tile grid

Our implementation utilizes CoreData to locally store and hold changes in our application. The **Persistence.swift** file contains code that loads and saves CoreData on application start and exit. **Tile.swift** and **Photo.swift** serve as objects for the CoreData. Each object contains public fields that are accessible within the page views to be displayed.

The files **ImagePicker.swift**, **PhotoCaptureView.swift**, **VoiceRecording.swift**, and **AudioPlayer.swift** are supporting files that perform the picture and audio functions. **ImagePicker** allows a picture to be selected from the user’s photo gallery; **PhotoCaptureView** allows a picture to be taken for a tile; **VoiceRecording** allows a custom voice memo to be recorded; and **AudioPlayer** creates instances where audio can be played from the device (this is for outputting the text-to-speech and custom voice memos). The **UtilityFunctions.swift** file contains supplementary functions which decode JSON files, copy files from the main bundle to the document directory, and get the image from the URL (detailed in the imageUrl field from the JSON).

ContentView.swift details the page structure of the application. See Figure 17. The app has 3 different pages: tiles, settings, and references. Each page’s code is detailed in their respective folders in Tab Bar Views (Speech, Settings, Home).

```swift
import SwiftUI
import CoreData

struct ContentView: View {
    var body: some View {
        TabView {
            SpeechGrid()
                .tabItem {
                    Image(systemName: "square.grid.3x2.fill")
                    Text("Tiles")
                }
            Settings()
                .tabItem {
                    Image(systemName: "gear")
                    Text("Settings")
                }
            References()
                .tabItem {
                    Image(systemName: "menucard.fill")
                    Text("References")
                }
        }
    }
}
```

Figure 17: ContentView.swift

The **WordsData.json** file contains the initial tile data. For each tile, the JSON file contains the fields **id**, **name**, **imageUrl**, **color**, and **inGrid**. The **id** is an identifier for each tile; the **name** field denotes the tile's word; the **imageUrl** denotes the filename for the tile's photo; the **color** denotes the tile color; the **inGrid** field denotes whether the tile is in the main grid or if it is in the frequented tile section. **WordsData.swift** parses the JSON and stores each tile's data into a word object detailed in **WordStruct.swift**.

The **SpeechGrid.swift** file contains the main functionality of our application. As seen in Figure 18, this file utilizes multiple fields for the view.

```swift
struct SpeechGrid: View {

    // ❎ CoreData managedObjectContext reference
    @Environment(\.managedObjectContext) var managedObjectContext

    // ❎ CoreData FetchRequest returning all tiles in the database
    @FetchRequest(fetchRequest: Tile.allTilesFetchRequest()) var allTiles: FetchedResults<Tile>

    //@EnvironmentObject var userData: UserData

    let columns = [ GridItem(.adaptive(minimum: 100), spacing: 20) ]

    @State private var sentence = ""
    @State private var isEdit = false
    @EnvironmentObject var audioPlayer: AudioPlayer

    // Create an utterance.
    let utterance = AVSpeechUtterance(string: "")

    // Create a speech synthesizer.
    let synthesizer = AVSpeechSynthesizer()

    func speak(_ utterance: AVSpeechUtterance) {
        let utterance = AVSpeechUtterance(string: sentence)
            self.synthesizer.speak(utterance)
        }

```

Figure 18: SpeechGrid.swift

The variable **allTiles** fetches all the tiles in the database and loads the grid with those tiles. The @State qualifier establishes that the field is subject to change in the view. The **sentence** field keeps track of the words being clicked, and will hold its value until the “Clear” button is pressed. The **isEdit** field keeps track of whether the user is currently editing a tile or not. The variables **utterance** and **synthesizer** are used for the function **speak** which is used with the “Speak” button. The **sentence** field is passed into **utterance**, which is spoken through **synthesizer**.

The grid is comprised of a VStack of 3 elements. Within the VStack, the first element is a HStack of the “Speak”, sentence preview bar, and “Clear" buttons. The second element is a vertical ScrollView of a LazyVGrid which contains the regular tiles in a 10 x 5 grid (10 x 4 if the frequently used tiles section is active). The third element is a horizontal ScrollView that displays the frequently used tiles if applicable. Within the second and third elements, the appropriate tiles are populated, with each tile acting as a button. Upon click, the respective word will be spoken through **synthesizer** and will be added to the **sentence** field.

The file AddTile.swift details the functionality of adding custom tiles. A snippet of the code is shown in Figure 19. We utilize many @State fields to hold the new tile’s values. The **showTileAddedAlert** and **showInputDataMissingAlert** are booleans that activate their

respective alerts when their conditions are met. The **recordingVoice** field is true if a custom voice memo is recorded; if false, the word will use the default text-to-speech synthesizer to output the word. **photoTakeOrPickChoices** details the two options for choosing a picture. The fields **showImagePicker**, **photoImageData**, and **photoTakeOrPickIndex** are used to store photo data and information about the photo selection process. The tile's word is stored in **word**, and if it is a frequently used tile the flag **frequentWord** will be true. **colorIndex**, **colorChoices**, and **colorStorage** detail the different colors that a tile can be. By default, the **colorIndex** is 2 which selects yellow. To add more colors, add the color and its UIColor value to **colorChoices** and **colorStorage,** respectively, and the option will appear in the picker.

```
15  struct AddTile: View {
16
17      @Environment(\.presentationMode) var presentationMode
18
19      @Environment(\.managedObjectContext) var managedObjectContext
20
21      @State private var showTileAddedAlert = false
22      @State private var showInputDataMissingAlert = false
23
24      @State private var recordingVoice = false
25
26      var photoTakeOrPickChoices = ["Camera", "Photo Library"]
27      @State private var showImagePicker = false
28      @State private var photoImageData: Data? = nil
29      @State private var photoTakeOrPickIndex = 1
30
31      @State private var word = "" //
32      @State private var frequentWord = false
33
34      @State private var colorIndex = 2
35      let colorChoices = ["Blue", "Green", "Yellow"]
36      let colorStorage = [UIColor.blue, UIColor.green, UIColor.yellow]
37
```

Figure 19: AddTile.swift

The file **EditTile.swift** has very similar code to that of **AddTile.swift**. See Figure 20 for a snippet of the code. This snippet displays the **navigationBarItems** of the **EditTile** view, which details the "Delete" and "Save" button.

```
112             .navigationBarTitle(Text("Edit Tile"), displayMode: .inline)
113             .navigationBarItems(
114                 leading:
115                     Button(action: {
116 //                        showTileDeleted = true
117                         managedObjectContext.delete(currTile)
118                     }) {
119                         Text("Delete")
120                     },
121                 trailing:
122                     Button(action: {
123                         if inputDataValidated() {
124                             saveTile()
125                             showTileEditedAlert = true
126                         } else {
127                             showInputDataMissingAlert = true
128                         }
129                     }) {
130                         Text("Save")
131                     })
132             .alert(isPresented: $showTileDeleted, content: { tileDeleted })
133         }
```

Figure 20: EditTile.swift

For each button on press, an alert is displayed to the user to signify the action's success. Deleting the tile changes **showTileDeleted** value to true, which activates its respective alert. The tile is then deleted from **managedObjectContext** which holds the CoreData objects. Attempting to save the tile will check the **inputDataValidated()** function, which makes sure that the necessary fields are filled (the word field). If it is, the **saveTile()** function is called, which updates the tile's fields based on the user inputs. Then the **showTileEditedAlert** is changed to true which activates the respective alert. If **inputDataValidated()** does not pass, the **showInputDataMissingAlert** field changes to true which will activate an alert that notifies the user that some necessary input data is missing.

8. Lessons Learned

Throughout the course of this semester, we faced many shortcomings, difficulties, technical issues, etc. However, through it all, we learned many new features of the Swift language that previous personal projects hadn't shown us before, and most importantly, the need for effective communication and vision. As many of our team members struggled with personal matters through the semester, this need for compensating for where someone could not accomplish a task at the moment was a valuable skill developed for each team member. Another lesson learned was understanding the desires of a client, and delivering the exact specifications they listed for us in the most optimal and concise manner. Through creating milestones for each task and organized planning to complete them, we were able to bring prepared versions of our application to each meeting with our client.

Overall, this project was eye opening to say the least. We were fortunate to have worked on a project that allowed us to give back to a certain group of users who are in need of non-verbal communicative assistance. The most important aspect of design that we came to learn was usability. When developing this application, we kept in mind the different features that could be implemented to allow for greater customizability, so that those who need to adjust certain settings would be able to do so. It was more so the desire to make things as easy as possible to handle for those who will be handling it that taught us that design is key in any given software. Through effective communication, teamwork, vision, and drive, our team was able to effectively create an application that our client should be pleased with.

9. Future Work

There is always room for improvement and refinement. Thus, there is more work that lies ahead. Although we accomplished what we set out to achieve this semester, upon talking with our client Mr. Newton, we saw that there was even more potential for this project. However, that work involved implementing features that didn't quite fit the scope of the semester after spending more than half designing and building it up from scratch. We do hope that this project is furthered by future teams which would enhance the target-user experience. Some of the functionality discussed as future work is described below:

- Predictive tile suggestion
    - A feature that can allow for the prediction of the next word based upon a user's input of tiles
    - When one tile is clicked, a specific subset of potentially relevant existing tiles is shown/highlighted compared to other tiles
    - Potentially using Hidden Markov Model
- Alternate voices
    - Younger/older
    - Different accents
    - Supporting different languages
- Testing
    - Present a target user with an iPad loaded with the FreeSpeech4VT application
    - Get feedback from
        - The caretaker
            - Ease of use
            - Ease of setup
            - Suggestions for improvement
        - The non-verbal user
            - Overall ergonomics
            - Anything glaringly difficult to understand or use
            - Suggestions for improvement
    - This section would entail obtaining an iPad from Virginia Tech, loading our application beta version, and giving it to a caretaker and non-verbal communicator
- Release to the public
    - Take the full and final product
    - Push onto the Apple App Store for the public to use

10. Acknowledgements

We would like to acknowledge the expertise and drive of our client, Mr. Matthew Newton. Mr. Matthew Newton provided our team with the equipment, information, and vision to carry out this project. We would also like to thank Dr. Edward A. Fox, who provided us with the connection to our client, as well as resources and connections to potential research participants.

• Mr. Matthew Newton
Department of Assistive and Education Technology
Email: matthewn@vt.edu

• Dr. Edward A. Fox
Office: 2160G Torgersen Hall
Phone: (540) 231-5113
Email: fox@vt.edu
Address: Dept. of CS, 1160 Torgersen Hall, Mail Code 0106, Virginia Tech, Blacksburg, VA 24061

11. References

[1]: U.S. Department of Justice. (2020, February). *A Guide to Disability Rights Laws*. ADA.gov. Retrieved April 12, 2022, from https://www.ada.gov/cguide.htm#anchor62335

[2]: U.S. Department of Health and Human Services. (2016, May 19). *Quick statistics about voice, speech, language*. National Institute of Deafness and Other Communication Disorders. Retrieved April 12, 2022, from https://www.nidcd.nih.gov/health/statistics/quick-statistics-voice-speech-language

[3]: *Disability Impacts All of Us*. Centers for Disease Control and Prevention. (2020, September 16). Retrieved April 12, 2022, from https://www.cdc.gov/ncbddd/disabilityandhealth/infographic-disability-impacts-all.html

[4]: Calder, A., & Townsend, B. (2021, December 24). Freespeech. Retrieved April 25, 2022, from https://app.freespeechaac.com/

[5]: Oh, S. (2022, February, 15). *Samueljoh00/FreespeechVT*. GitHub. Retrieved April 25, 2022, from https://github.com/samueljoh00/FreespeechVT

[6]: *2.1 Git Basics - Getting a Git Repository*. Git. (2021, November 11). Retrieved April 25, 2022, from https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

[7]: *Proloquo2Go - AAC app with symbols*. AssistiveWare. (2022). Retrieved April 25, 2022, from https://www.assistiveware.com/products/proloquo2go