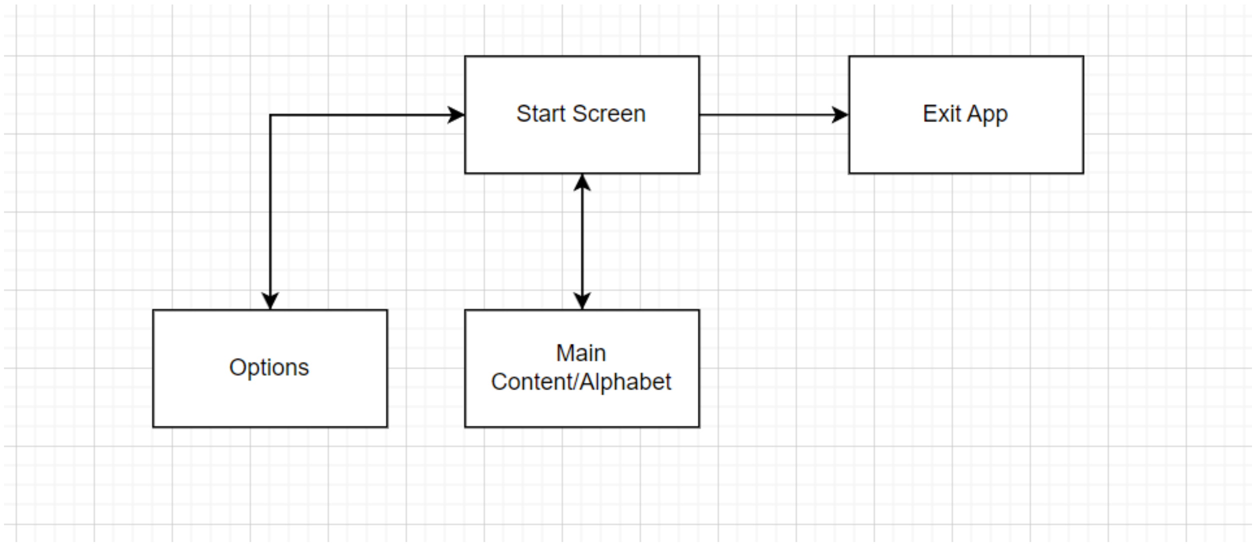
Purpose and User requirements of the app

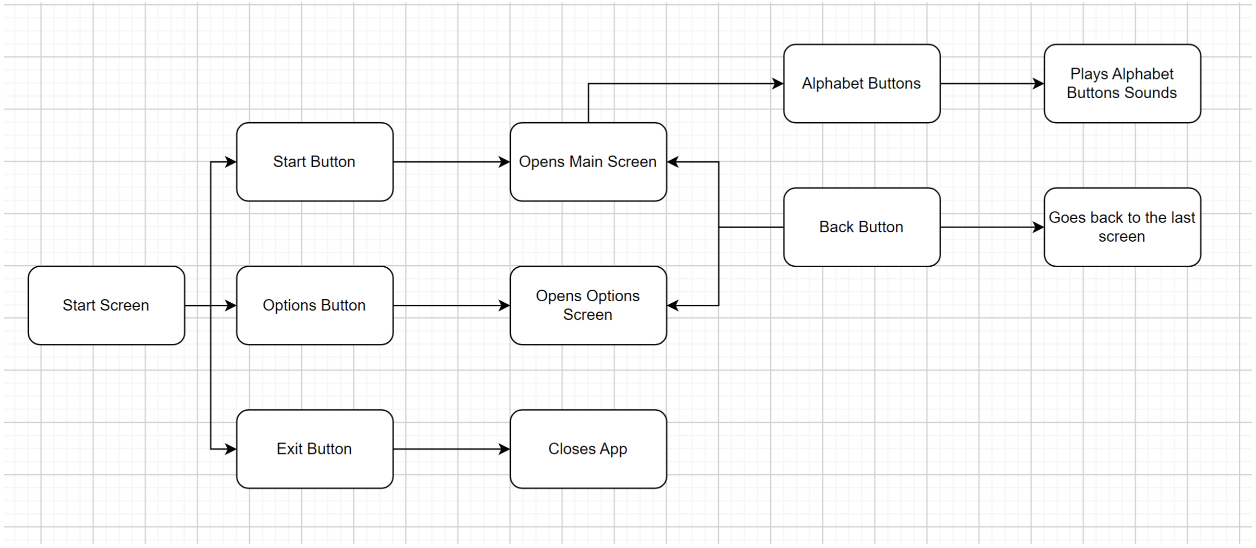
The purpose of the app is to help children learn the alphabet. The user requirements would be a mobile phone and an internet connection to download the app with enough storage space available.

Flowchart



Control Structure

My app starts with three options: Start, options and exit. Pressing start takes you to another screen with buttons for all the letters of the alphabet. Each button when pressed will pronounce the letter. There is also a back button on the alphabet screen which takes you back to the start screen. The option button at the start screen takes you to another screen for options. The options screen also has a back button which takes you back to the start screen. The exit button closes the application.



Pseudocode, Events and Error Handling

The main tasks are displaying the Start screen, the Alphabet screen, and the Options screen, as well as handling user inputs and providing error handling.

```
Display the Start screen
Wait for the user to select an option
If the user selects the "Start" option:
   Display the Alphabet screen with buttons for each letter of the alphabet
   Wait for the user to press a button
If the user presses a letter button:
   Pronounce the corresponding letter
If the user presses the "Back" button:
      Display the Start screen
If the user selects the "Options" option:
   Display the Options screen
   Wait for the user to customize settings
If the user moves the volume slider:
    Slider position changes
    Volume label changes
If the user presses the dark mode switch:
    Background colour changes to black
    Label colour changes to white
If the user presses the "Back" button:
      Display the Start screen
If the user selects the "Exit" option:
   Close the application
If an error occurs while handling an event:
   Display an error message and provide a fallback option
```

Data structure

alphabet = ["A", "B", "C", "D", "E", "F","G", "H", "I", "J", "K", "L","M", "N", "O", "P", "Q", "R","S", "T", "U", "V", "W", "X","Y", "Z"]

Data input and output:

Read the data for alphabet and pronunciations from a WAV file.

Display the alphabet buttons on the screen.

When a letter button is pressed, retrieve its corresponding pronunciation from the pronunciations data structure stored in a TinyDB and play the audio.

When the slider for the volume is changed, store the value in a TinyDB and display the change.

Data manipulation:

Sort the alphabet list in alphabetical order.

Group the letters by their pronunciation.

Error handling:

Display an error message if the data retrieval or manipulation fails.

Reporting

If the user presses the back button, return to the previous screen.

If the user presses the exit button, display a confirmation message that says, "Are you sure you want to exit?" If the user confirms, close the application.

If the user presses a letter button, play its corresponding pronunciation.

Main program tasks

At the top of the screen is the title of the app “Learn The Alphabet” and below are three buttons at the center of the screen: Start, options and exit and are in a vertical arrangement.

Each of the letters of the alphabet are arranged horizontally in four rows. Below the letters is the back button that goes back to the start screen.

The option screen has a slider for controlling the volume of the letters with a label next to it that displays the current volume at the top of the screen and a back button underneath.

Descriptions of the method of solution

My app is designed with buttons that are labelled with a letter and when they are pressed, they play a sound file pronouncing the letter. The volume of the sound file can be adjusted in the options menu. The app doesn’t have anything for accessibility. The app only has one method of teaching.

A list of any pre-defined programs/code snippets (including any functions or sub-routines)

Screen 1

When button 1 click – open another screen – Screen 2

When button 2 click – open another screen – Screen Options

When button 3 click – close application

Screen 2

When button A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, R, S, T, U, V, W, X, Y, Z click – call player A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, R, S, T, U, V, W, X, Y, Z start

When button back click – open another screen – Screen 1

When Screen 2 initialize – Set player A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, R, S, T, U, V, W, X, Y, Z volume to value Volume

Screen Options

When Screen Options initialize – Set label1 text to value with tag CurrentVolume – Set slider position to value with tag Volume

Initialize global CurrentVolume to blank text

Initialize global CurrentVolumeS to blank text

When Slider position changed – Set label1 text to thumb position – store value thumb position with tag Volume – store value label1 text with tag CurrentVolume – Set global CurrentVolume to label text – Set global CurrentVolumeS to thumb position

Source table

Asset	Source
Labels	App Inventor
Buttons	App Inventor
Slider	App Inventor
Wav sound files	www.soundspunos.com

Test Plan

Test	Expected Result	Actual Result
Does button1 in screen1 work?	The app opens to screen 2	The app opens to screen 2
Does button2 in screen1 work?	The options screen opens	The options screen opens
Does button3 in screen1 work?	The app closes	The app closes
Does the buttons for the sound files work?	The buttons play the sound files	The buttons play the sound files
Does the back button in the alphabet screen work?	The back button works	The back button works
Does the slider for the volume in the options screen work?	The slider works	The slider works
Does the back button in the options screen work?	The back button works	The back button works

A brief outline of any alternative solutions for the intended software program and why these alternative designs were rejected

An alternative solution would be writing in visual code to make the app, however this was rejected as it would take more time and would require more research. App inventor is a much easier alternative as it has pre-built code for everything with a simple interface.

Design justification that describes how the chosen design fulfils the purpose and user requirements and how the app is suitable for the end users

The app is designed to help children learn the alphabet and requires a mobile phone with an internet connection to download the app and sufficient storage space. The design I have chosen has a simple interface with few features so children can easily navigate the app and so it doesn't distract from the purpose of teaching them the alphabet. I chose to made buttons for each of the letters of the alphabet and when one of the buttons is pressed, a sound file will play pronouncing the letter so they can see which letter corresponds with the sound, which makes learning the alphabet easier. The volume slider will allow children to adjust the volume of the sounds, so they can hear the pronunciation louder and more clearly which will help them learn the alphabet more easily. The chosen control structure, pseudocode, and events contribute to the overall usability of the app by being simple and easy to use so children don't have any trouble using the app.

Demonstrate the quality and thoroughness of the design work

Test	**Expected Result**	**Actual Result**
Can you play all the sounds?	All the sounds play	All the sounds play
Does turning off the screen with the app open, break the app?	The app doesn't break	The app doesn't break

A review of the design in light of any constraints, such as variable screen sizes

I have designed this mobile app to be a simple educational tool for children to learn the alphabet. To do this, I have included features such as a simple and intuitive control structure that allows users to easily navigate through the app.

One constraint that I have kept in mind while designing this app is the variability of screen sizes on different mobile devices. I have tried to make sure that the layout and placement of elements are such that they are easily accessible and visible on screens of different sizes. For example, I have made sure that the alphabet buttons are arranged horizontally in four rows, making them easily viewable on both small and large screens.

To ensure the smooth running of the app, I have included several pseudocode events and error handling features. These include displaying error messages if there is a problem with data retrieval or manipulation and providing fall-back options for the user in such cases.

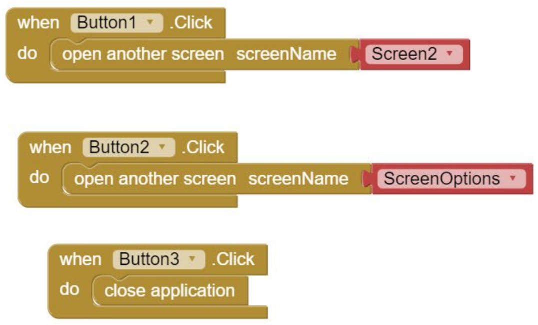
The data structure of the app includes a list of alphabet letters, which is sorted alphabetically and grouped by pronunciation. Additionally, the app reads data from a WAV file and displays the alphabet buttons on the screen. When a letter button is pressed, the corresponding pronunciation is retrieved from the pronunciations data structure and played.

To make the app more user-friendly, I have included a slider in the options screen to allow users to adjust the volume of the pronunciations. The slider position is displayed in a label, and changes to the slider position are stored and retrieved using tags.

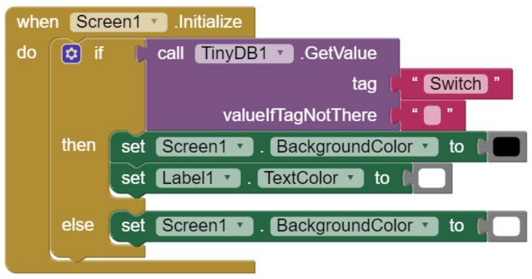
I am pleased with the design of this app, and I am confident that it will be a valuable tool for children learning the alphabet.

Code Annotations

Screen 1



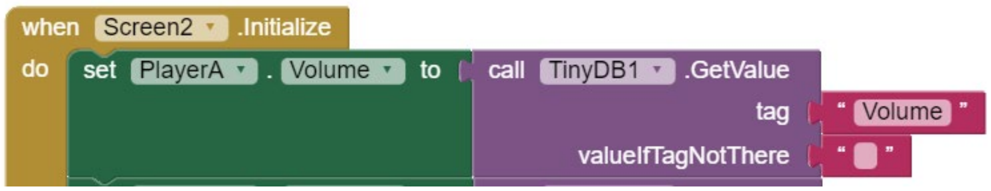
```
when Button1 .Click
do  open another screen  screenName  Screen2

when Button2 .Click
do  open another screen  screenName  ScreenOptions

when Button3 .Click
do  close application
```



```
when Screen1 .Initialize
do  if    call TinyDB1 .GetValue
                       tag  " Switch "
          valueIfTagNotThere  "  "
    then  set Screen1 . BackgroundColor to
          set Label1 . TextColor to
    else  set Screen1 . BackgroundColor to
```

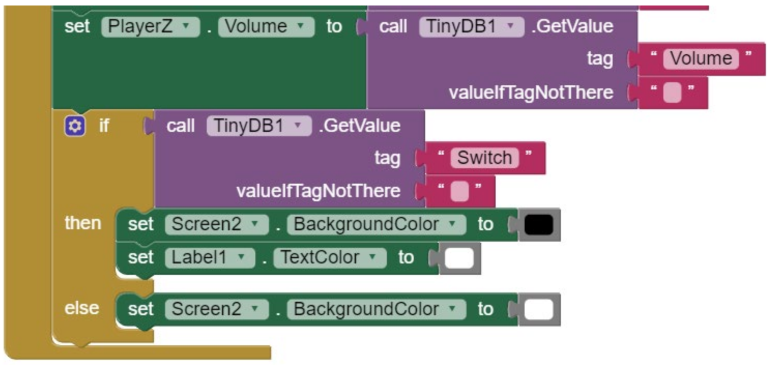
Firstly, when screen1 starts up or when you go back to screen1 from another screen it will run the “when screen1.initialize” function, which checks if dark mode has been turned on or off and changes the colours if dark mode has been changed. “when button1.click” is when the button is pressed, it will open another screen called Screen2. “when button2.click” is when the button is pressed, it will open another screen called ScreenOptions. “when button2.click” is when the button is pressed, it will close the app.

Screen2



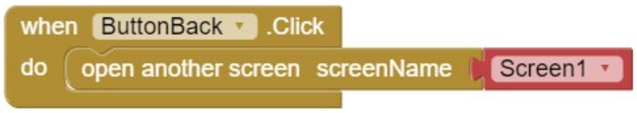
```
when Screen2 .Initialize
do  set PlayerA . Volume to  call TinyDB1 .GetValue
                                          tag  " Volume "
                             valueIfTagNotThere  "  "
```

When screen2 opens it will run the “when screen1.initialize” function, which will get the value that the volume has been set to in the options screen and change the volume of the players to that value. This is repeated for all the letters.

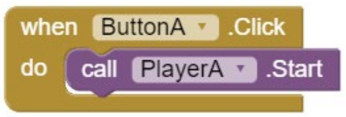


```
set PlayerZ . Volume to call TinyDB1 .GetValue
                                 tag " Volume "
                                 valueIfTagNotThere " "
if call TinyDB1 .GetValue
                 tag " Switch "
                 valueIfTagNotThere " "
then set Screen2 . BackgroundColor to
     set Label1 . TextColor to
else set Screen2 . BackgroundColor to
```

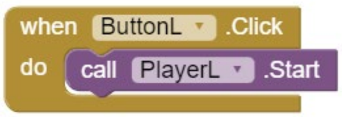
Also, in the screen2 “when screen1.initialize” function is code that checks if dark mode has been turned on or off and changes the colours if dark mode has been changed.



```
when ButtonBack .Click
do   open another screen screenName Screen1
```



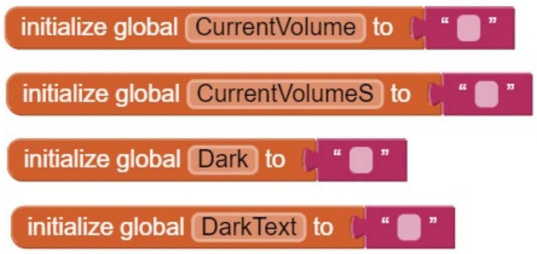
```
when ButtonA .Click
do   call PlayerA .Start
```



```
when ButtonL .Click
do   call PlayerL .Start
```

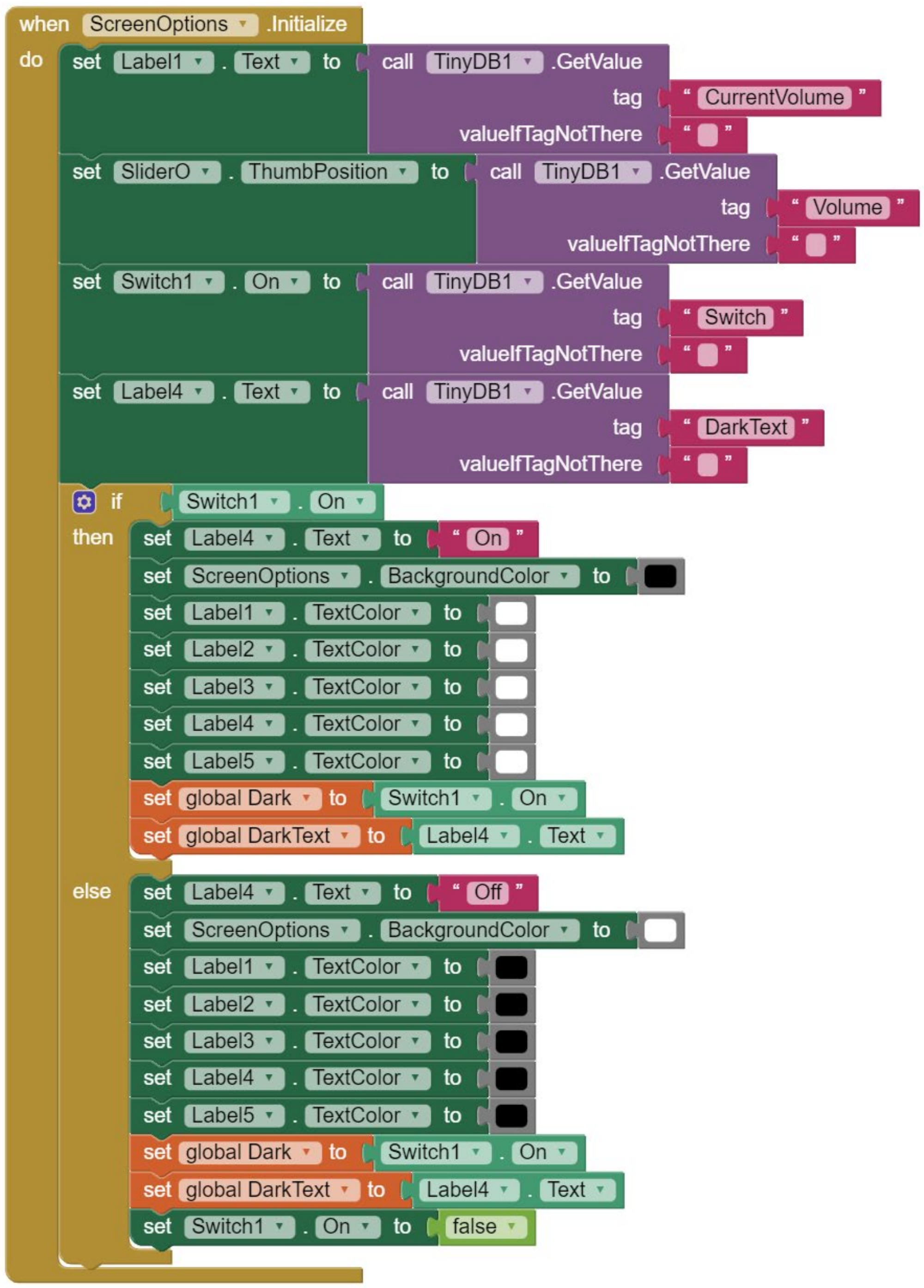
“when ButtonBack.click” is when the button is pressed, it will open another screen called Screen1. “when ButtonA.click” is when the button is pressed, it will play the sound attached to playerA. This is done for all the letters of the alphabet.

ScreenOptions



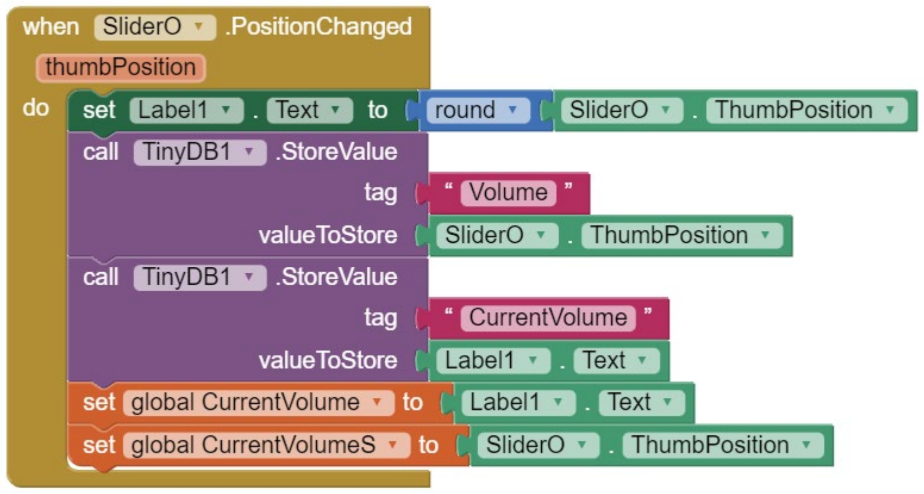
```
initialize global CurrentVolume to " "
initialize global CurrentVolumeS to " "
initialize global Dark to " "
initialize global DarkText to " "
```

This code declares empty global variables, so they do nothing until something is assigned.



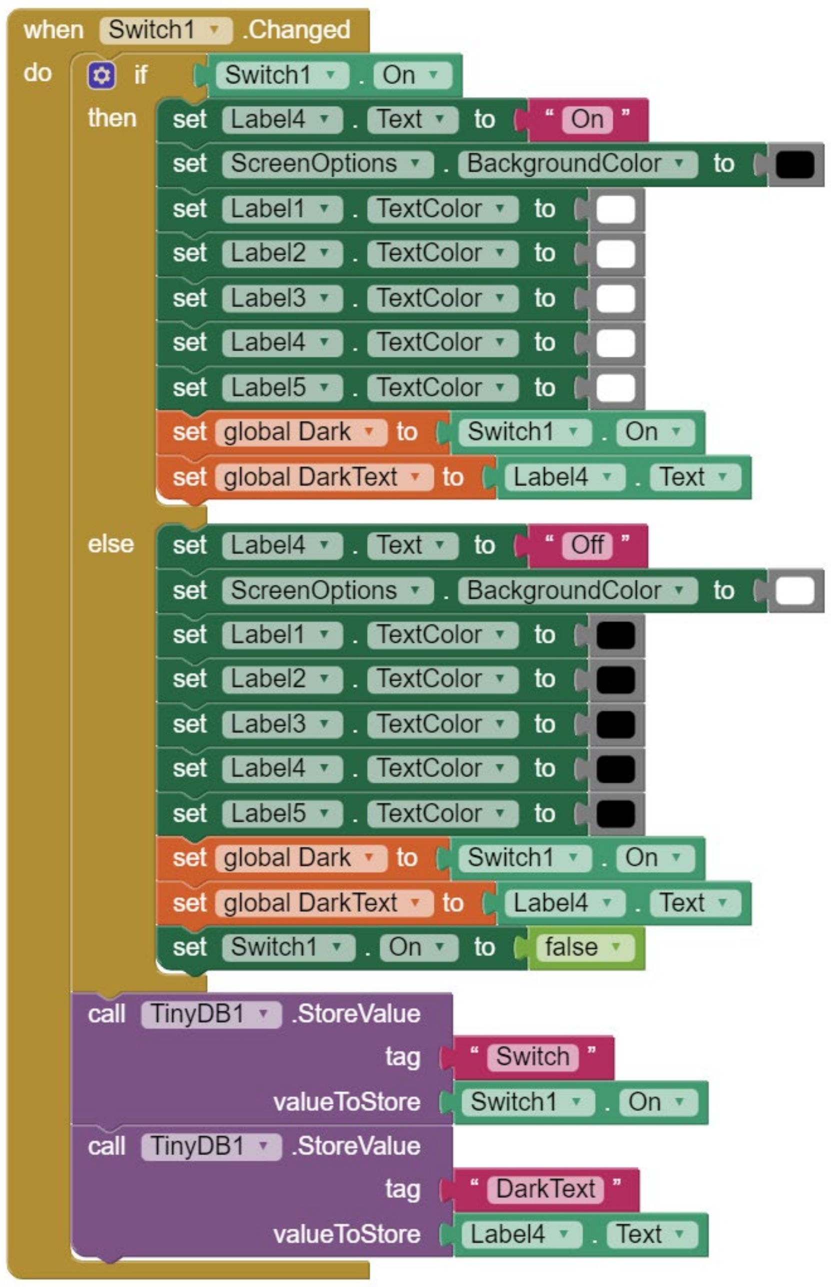
```
when ScreenOptions .Initialize
do  set Label1 . Text to  call TinyDB1 .GetValue
                                         tag " CurrentVolume "
                           valueIfTagNotThere " "
    set SliderO . ThumbPosition to  call TinyDB1 .GetValue
                                                  tag " Volume "
                                    valueIfTagNotThere " "
    set Switch1 . On to  call TinyDB1 .GetValue
                                       tag " Switch "
                         valueIfTagNotThere " "
    set Label4 . Text to  call TinyDB1 .GetValue
                                        tag " DarkText "
                          valueIfTagNotThere " "
    if  Switch1 . On
    then  set Label4 . Text to " On "
          set ScreenOptions . BackgroundColor to ⬛
          set Label1 . TextColor to ⬜
          set Label2 . TextColor to ⬜
          set Label3 . TextColor to ⬜
          set Label4 . TextColor to ⬜
          set Label5 . TextColor to ⬜
          set global Dark to  Switch1 . On
          set global DarkText to  Label4 . Text
    else  set Label4 . Text to " Off "
          set ScreenOptions . BackgroundColor to ⬜
          set Label1 . TextColor to ⬛
          set Label2 . TextColor to ⬛
          set Label3 . TextColor to ⬛
          set Label4 . TextColor to ⬛
          set Label5 . TextColor to ⬛
          set global Dark to  Switch1 . On
          set global DarkText to  Label4 . Text
          set Switch1 . On to  false
```

The “when screen1.initialize” function is code that sets all the labels and positions of the slider and switch to what has already been set when you reopen the screen. The “if switch” function is for dark mode, so if dark mode was turned on or off, it will change according to what was previously chosen.



```
when SliderO .PositionChanged
  thumbPosition
do  set Label1 . Text to  round  SliderO . ThumbPosition
    call TinyDB1 .StoreValue
                      tag  " Volume "
             valueToStore  SliderO . ThumbPosition
    call TinyDB1 .StoreValue
                      tag  " CurrentVolume "
             valueToStore  Label1 . Text
    set global CurrentVolume to  Label1 . Text
    set global CurrentVolumeS to  SliderO . ThumbPosition
```

This function is a slider for the volume, which changes the number on the label to what has been selected, stores the values for the position on the slider and number on the label and sets the empty global variables to those values.



```
when Switch1 .Changed
do  if  Switch1 . On
    then  set Label4 . Text to " On "
          set ScreenOptions . BackgroundColor to ⬛
          set Label1 . TextColor to ⬜
          set Label2 . TextColor to ⬜
          set Label3 . TextColor to ⬜
          set Label4 . TextColor to ⬜
          set Label5 . TextColor to ⬜
          set global Dark to Switch1 . On
          set global DarkText to Label4 . Text
    else  set Label4 . Text to " Off "
          set ScreenOptions . BackgroundColor to ⬜
          set Label1 . TextColor to ⬛
          set Label2 . TextColor to ⬛
          set Label3 . TextColor to ⬛
          set Label4 . TextColor to ⬛
          set Label5 . TextColor to ⬛
          set global Dark to Switch1 . On
          set global DarkText to Label4 . Text
          set Switch1 . On to false
    call TinyDB1 .StoreValue
                  tag " Switch "
         valueToStore Switch1 . On
    call TinyDB1 .StoreValue
                  tag " DarkText "
         valueToStore Label4 . Text
```

This function is for dark mode, if the switch has been turned on, it will then change the label text to "On", background colour to black, label colours to white and set the empty global variables to the position of the switch and the label for "On" and "Off". If the switch has been turned off, it will then change the label text to "Off", background colour to white, label colours to black and set the empty global variables to the position of the switch and the label for "On" and "Off".

Comment on the quality of the code; for example, maintainability (how easily the code can be modified), portability (on different platforms) and usability.

The code was made with MIT app inventor which uses pre-defined in the form of blocks which makes modifying the code easy as it is simple to understand and there are many resources available that can help people learn how to use the code.

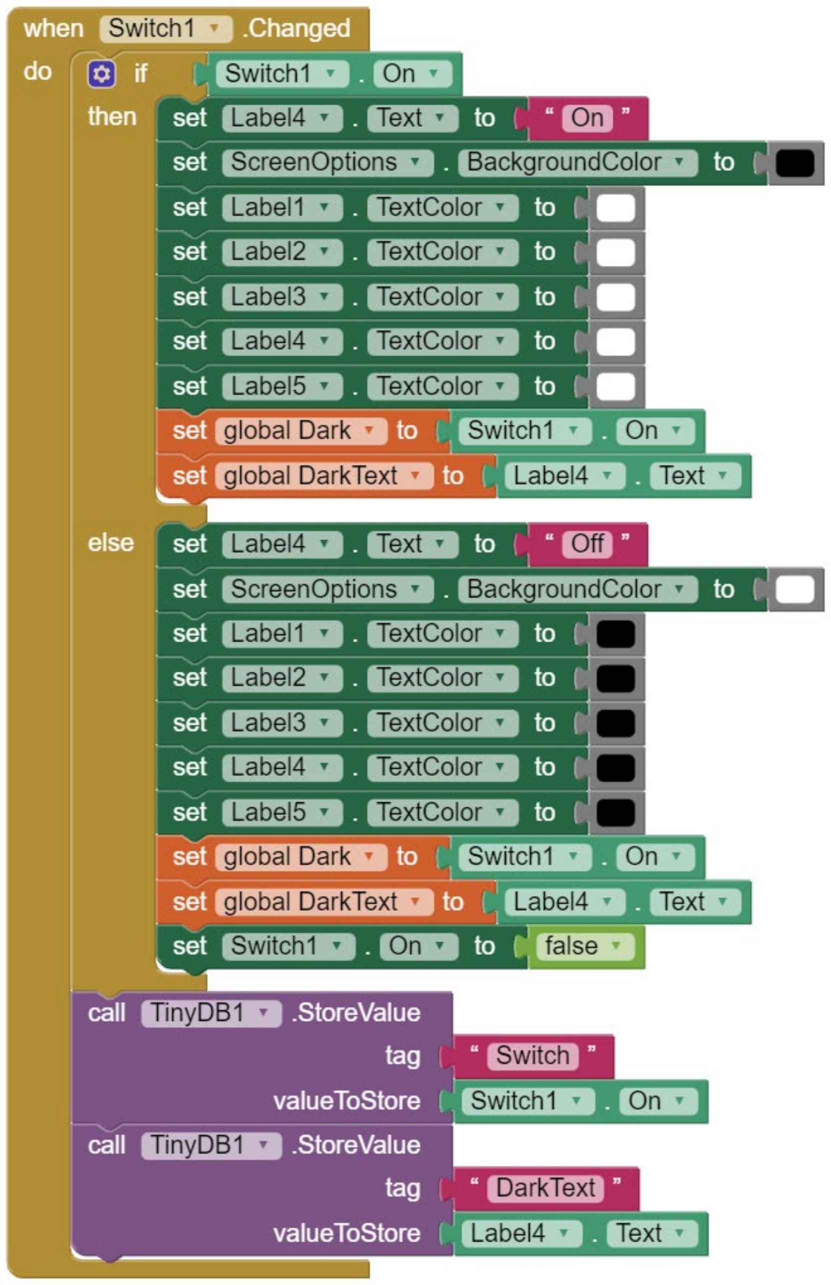
The app was made using MIT app inventor which makes portability easy as it has options available for different platforms.

The code is simple and easy to use as it is pre-defined code which makes it easy to understand and make sense of.

Questionnaire

Question	**Answer**
Does the app work?	Yes, the app works, all the buttons and functions are working as they are meant to be.
Is it easy to use?	Yes, the app is simple with not many features but it could include a dark mode which would make it easier to look at.
Does the app meet its intended purpose?	Yes, the app pronounces the letters in correspondence to the buttons and labels
What improvements could be made?	Adding a dark mode would be good for some people who may prefer it.

Improvements to app



```
when Switch1 .Changed
do  if Switch1 . On
    then set Label4 . Text to " On "
         set ScreenOptions . BackgroundColor to ■
         set Label1 . TextColor to □
         set Label2 . TextColor to □
         set Label3 . TextColor to □
         set Label4 . TextColor to □
         set Label5 . TextColor to □
         set global Dark to Switch1 . On
         set global DarkText to Label4 . Text
    else set Label4 . Text to " Off "
         set ScreenOptions . BackgroundColor to □
         set Label1 . TextColor to ■
         set Label2 . TextColor to ■
         set Label3 . TextColor to ■
         set Label4 . TextColor to ■
         set Label5 . TextColor to ■
         set global Dark to Switch1 . On
         set global DarkText to Label4 . Text
         set Switch1 . On to false
    call TinyDB1 .StoreValue
                         tag " Switch "
                valueToStore Switch1 . On
    call TinyDB1 .StoreValue
                         tag " DarkText "
                valueToStore Label4 . Text
```

I added a dark mode to the app so people who prefer it can switch.

Review

The app is suitable for the user requirements and target audience as the design has a simple interface with few features so children can easily navigate the app and so it doesn't distract from the purpose of teaching them the alphabet. The app is available on iOS and Android and does not need any account login to access so children can use the app.

The app is suitable for the intended purpose as the app is designed with buttons for each of the letters of the alphabet and when one of the buttons is pressed, a sound file will play pronouncing the letter so they can see which letter corresponds with the sound, so they can learn the alphabet by association. A volume slider will allow children to adjust the volume of the sounds, so they can hear the pronunciation louder and more clearly which will help them learn the alphabet more easily.

One constraint I had when using MIT app inventor was being limited to the pre-defined code available. Another constraint that I had while designing the app is the variability of screen sizes on different mobile devices. From the user feedback I added a dark mode which will make the app easier to look at for some children which will increase the range of children that can use the app. From the testing I did, I increased the size of the buttons to fill up more of the screen and make it more visible so the screen looked less empty. I also added some error-handling to issues I ran into to make the app easier to use.

The initial design was a simple interface with only a few features that has buttons for each of the letters of the alphabet and when one of the buttons is pressed, a sound file will play pronouncing the letter. The volume slider was to adjust the volume of the sounds, so they can hear the pronunciation louder and more clearly which will help them learn the alphabet more easily. The changes I made involved increasing the size of the buttons to fill up more of the screen and make it more visible so the screen looked less empty and adding a dark mode to the app which will make it easier to look at for some children increasing the range of children that would use the app.

One improvement for the completed app is adding a text box where children can type in the letter after hearing the pronunciation to see if they can get it right which will test their memory. Another improvement is adding word examples for each of the letters, so children can see how the letter is used in a word. The last improvement would be adding a simple game to the app were children can test their knowledge and help them learn while being fun.