Language Phrasebook Application

1. Analysis

The task of the assignment is to develop an iOS language phrasebook which could be used to record words or phrases in two languages.

Firstly, the application should be able to record words of both languages and some other relating attributes and notes. Each pair of words can also have an optional image and its copyright. Besides these, users can add tags, which are not predetermined by the application, to a pair of word and each pair of word can have zero or more tags. The tag will be used to classify or filter words or phrases. All of these information should be stored for retrieve. Because there will be a lot of words or phrases as the time goes on, a search function will be implemented so that users can search for words or phrases and find a word or phrase more easily. If there are more than one result, all of them should be displayed.

Secondly, a revision exercise should be provided to users for practicing. For users might want to review words to memorise them, there should be such a function to help them. And the function should have an appropriate method to retrieve words from stored data so that words would not be displayed in the same sequence every time. I have an English word book in which the first word is abandon. The first thing the book tells me is abandon. And it is really awful to practice from the same place in the same sequence from A to Z.

Finally, a download function should be implemented for users would not like to manually type in words or phrases and the relating information always. It could be really monotonous to do that. The sets of word downloaded from the Internet is said to be in JSON format so the application should be able to download the JSON file in a given URL and parse the contents.

2. Design

As described in the former section, the task is divided into three main parts, therefore Tab Bar Controller would be used to manage different View Controllers.

To deal with the first part of the application, both words displaying function and edit function would be implemented. There will be two viewers of displaying functions displaying words. In one of the views, word pairs would be displayed in alphabetical order according to the first word of the pair(we call it wordone). If some pairs have the same wordone, they would be order by the second word of the pair(wordtwo). In another viewer, word pairs would be classified into several groups according to their tags. As they might have multiple tags, a word pair would appear under different groups of tags.

Another function, the edit function, of the first part, would take the job to edit a word pair(including adding a new word pair). The data would be stored in a SQLite database. Wordone and wordtwo would be combined to the composite primary key of the database to uniquely identify a word pair. There would also be seven attributes namely type, note, image, imageurl, imagecredit, imagelicense and tag contained in the database. Type is used to describe the property of wordone; note would store a user's annotation of the word pair; in image column, name of the image used should be entered, while information about its copyright should also be added to imagecredit column and imagelicense column if available and the source url of the image should be placed in imageurl column; the tag column would contain zero or more tags added by users. The edit function would check the input whether the input is nil or not. If the input is nil, a default value would be stored in database to prevent some null problems happening.

The revision function would have an method to pick word pair and display only the wordone on the screen. A button would be added to display the wordtwo and other information. Rather than giving out some word options to test the user, the application would provide two buttons, correct button and false button, to obtain the outcome of the user's answer after they press the button to see the wordtwo. There would also be a button named next allowing the user to move to the next word manually rather than automatically in some way, which is due to the reason that users might want to have a look at the word pair to remember the words.

The last part, the download function would be quite simple for we are just going to download JSON file from a given URL. So on the screen a text field would be displayed allowing users to type in an URL, while a button namely download would be placed behind the text filed to trigger download method.

The interaction between the user and the application is shown in Figure 2.1.

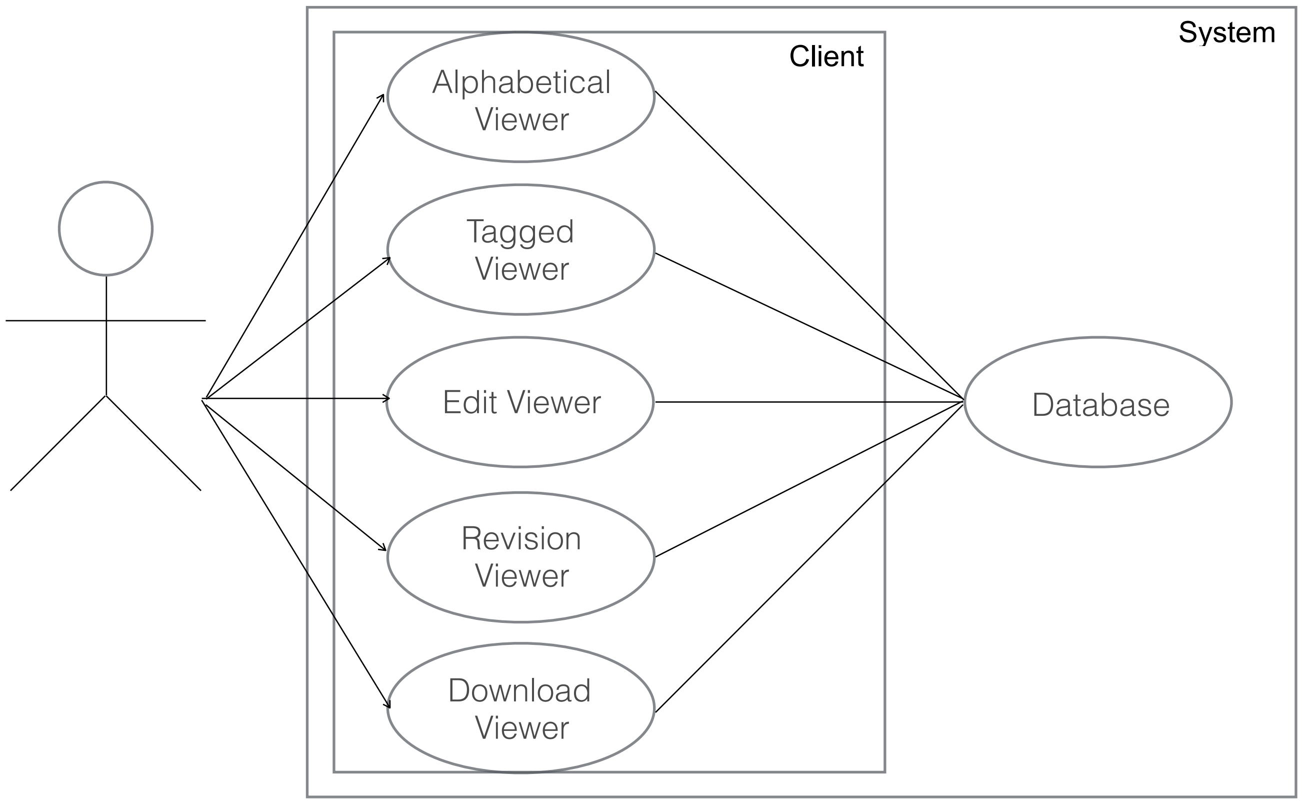


Figure 2.1

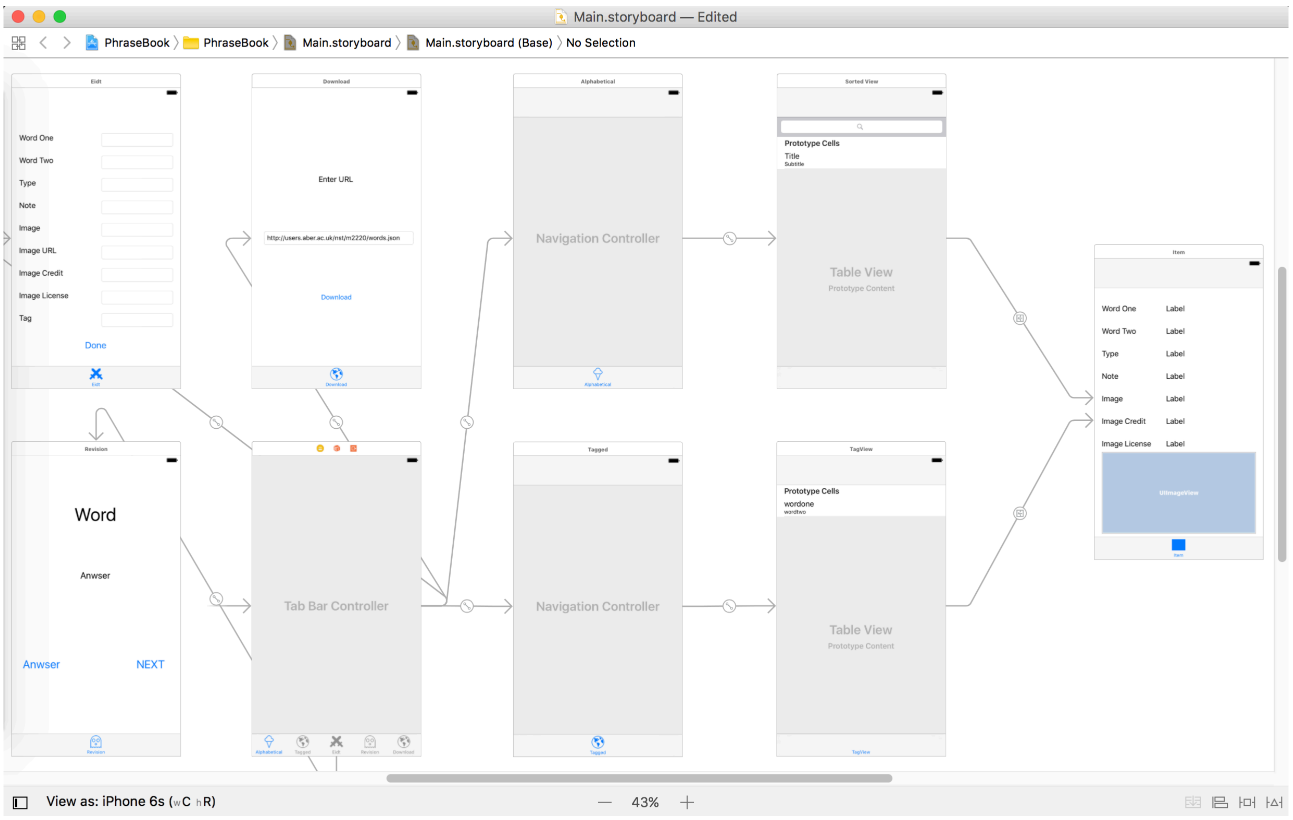
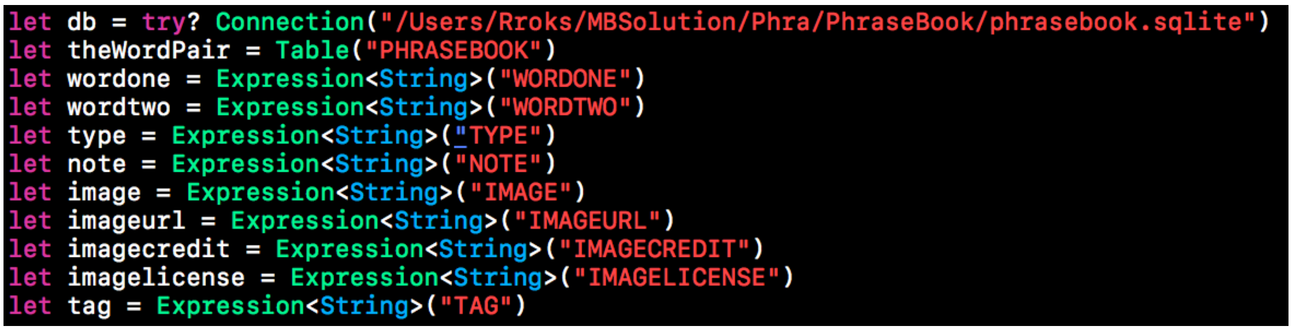


Figure 3.1

3. Implementation

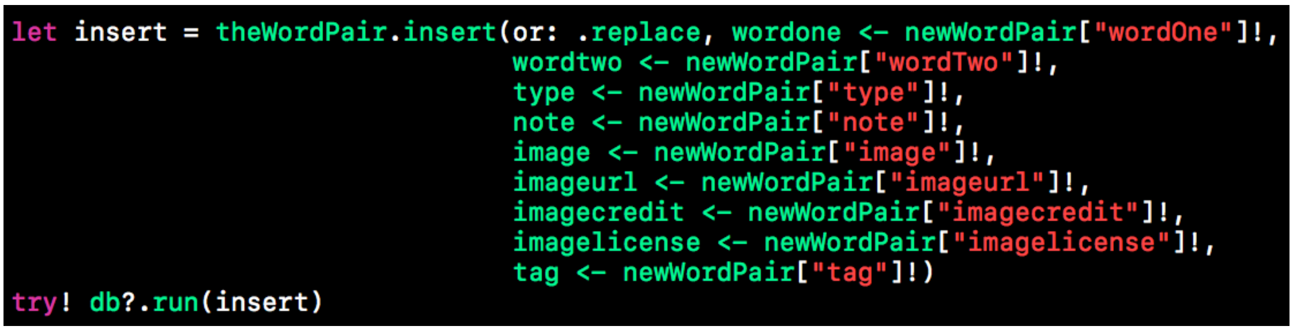
The overall view of the implementation is shown in Figure 3.1. To some reasons, the implementations didn't follow the original design strictly. There are five tabs namely Alphabetical, Tagged, Edit, Revision and Download. Both Alphabetical tab and Tagged tab have a Navigation Controller so users can move through a hierarchy of views. Meanwhile, both Alphabetical view and Tagged view are connected to a word pair displayer.

In Edit tab, night text files are implemented for obtaining input and a button named Done to trigger store method. A third party library called SQLite.swift is used here to assist dealing with SQLite compilation. Some global variables are also implemented for supporting the use of SQLite.swift as shown in Figure 3.2. There is a problem unsolved that the path to the database is static.The statements shown in Figure 3.3 make sure that the program will update the database rather than crash when the input word pair is already exist in the database.



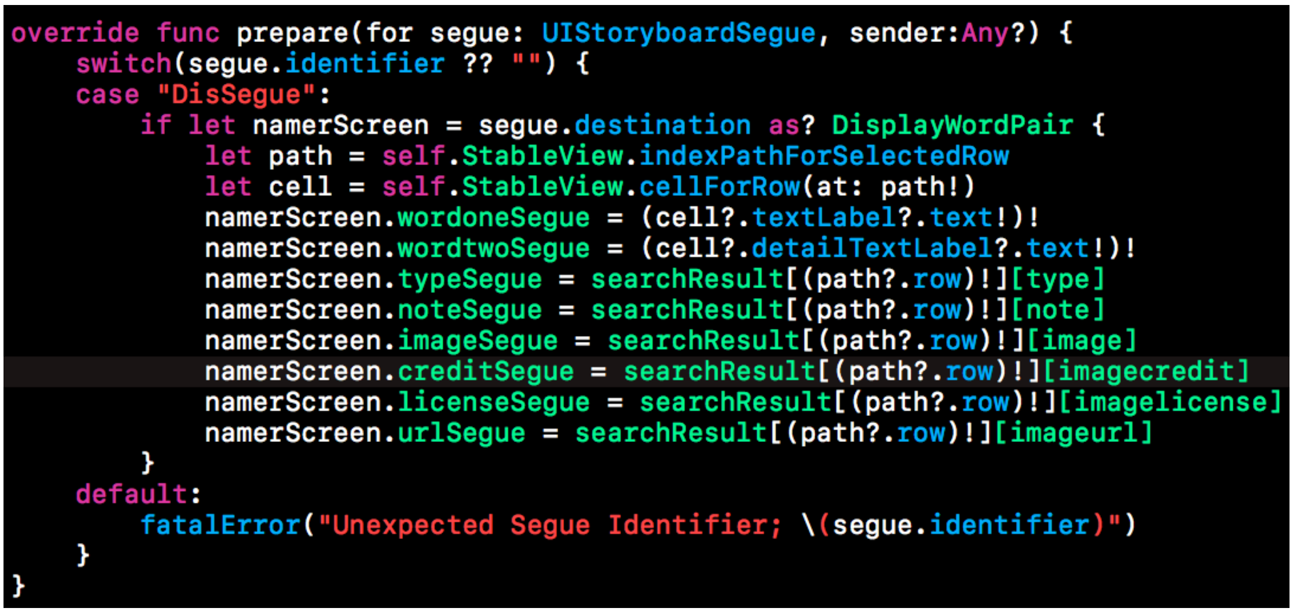
```
let db = try? Connection("/Users/Rroks/MBSolution/Phra/PhraseBook/phrasebook.sqlite")
let theWordPair = Table("PHRASEBOOK")
let wordone = Expression<String>("WORDONE")
let wordtwo = Expression<String>("WORDTWO")
let type = Expression<String>("TYPE")
let note = Expression<String>("NOTE")
let image = Expression<String>("IMAGE")
let imageurl = Expression<String>("IMAGEURL")
let imagecredit = Expression<String>("IMAGECREDIT")
let imagelicense = Expression<String>("IMAGELICENSE")
let tag = Expression<String>("TAG")
```

Figure 3.2



```
let insert = theWordPair.insert(or: .replace, wordone <- newWordPair["wordOne"]!,
                                wordtwo <- newWordPair["wordTwo"]!,
                                type <- newWordPair["type"]!,
                                note <- newWordPair["note"]!,
                                image <- newWordPair["image"]!,
                                imageurl <- newWordPair["imageurl"]!,
                                imagecredit <- newWordPair["imagecredit"]!,
                                imagelicense <- newWordPair["imagelicense"]!,
                                tag <- newWordPair["tag"]!)
try! db?.run(insert)
```

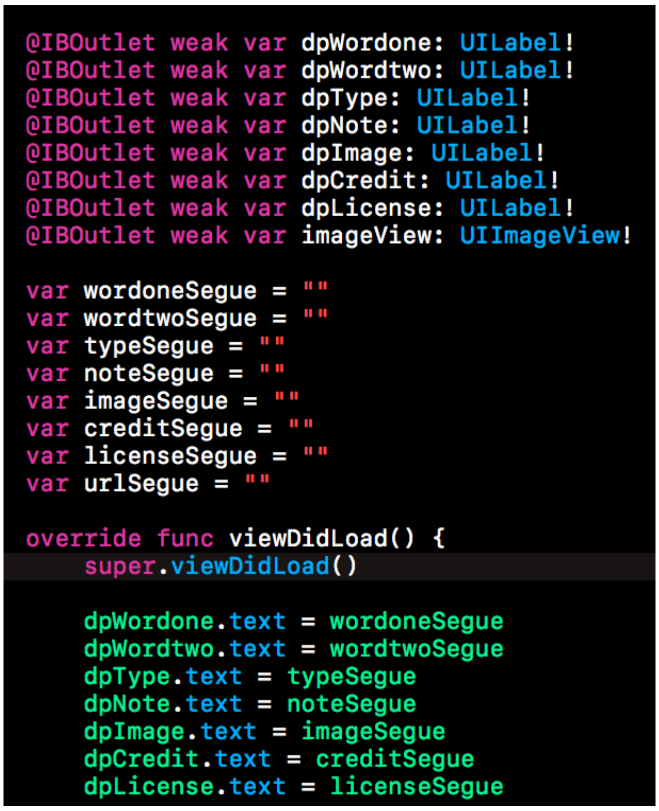
Figure 3.3



```
override func prepare(for segue: UIStoryboardSegue, sender:Any?) {
    switch(segue.identifier ?? "") {
    case "DisSegue":
        if let namerScreen = segue.destination as? DisplayWordPair {
            let path = self.StableView.indexPathForSelectedRow
            let cell = self.StableView.cellForRow(at: path!)
            namerScreen.wordoneSegue = (cell?.textLabel?.text!)!
            namerScreen.wordtwoSegue = (cell?.detailTextLabel?.text!)!
            namerScreen.typeSegue = searchResult[(path?.row)!][type]
            namerScreen.noteSegue = searchResult[(path?.row)!][note]
            namerScreen.imageSegue = searchResult[(path?.row)!][image]
            namerScreen.creditSegue = searchResult[(path?.row)!][imagecredit]
            namerScreen.licenseSegue = searchResult[(path?.row)!][imagelicense]
            namerScreen.urlSegue = searchResult[(path?.row)!][imageurl]
        }
    default:
        fatalError("Unexpected Segue Identifier; \(segue.identifier)")
    }
}
```

Figure 3.4

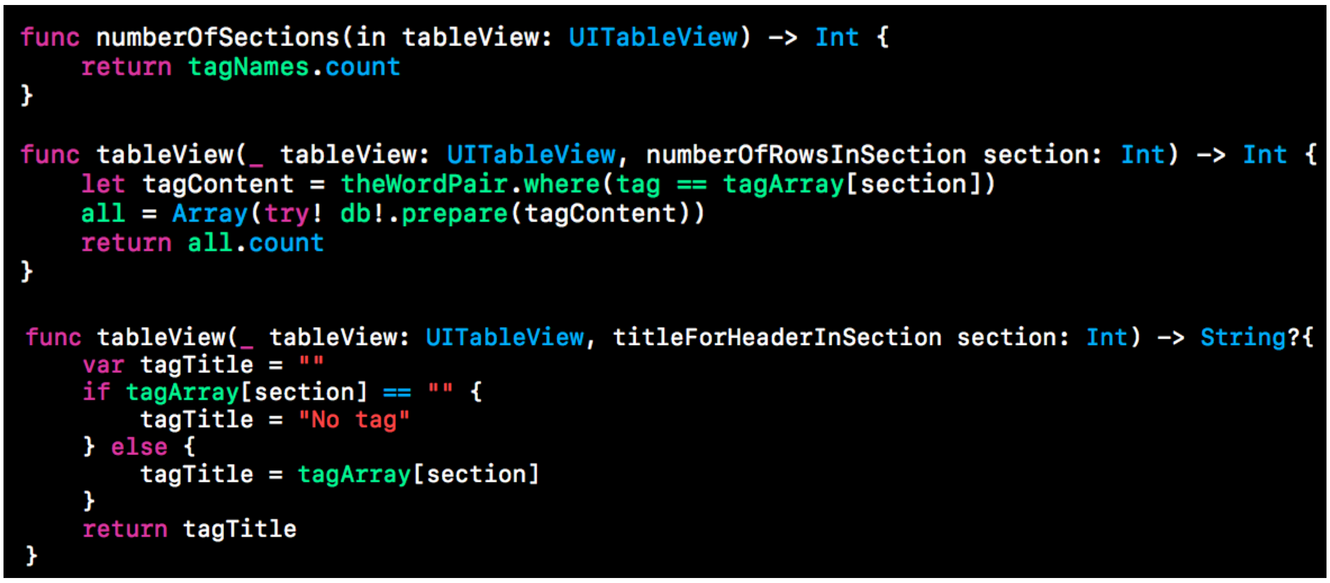
In Alphabetical view, a search bar is implemented. Users can search for a word one through it. Two arrays are implemented. One named all is used to retrieve all data from database, another one will change dynamically according to the user's option. To use the search bar, the protocol UISearchResultsUpdating is extended. When implementing the function prepare(for segue: UIStoryboardSegue, sender:Any?), an unwrapping problem was met. Some variables are used to help solve the problem. The codes in Alphabetical view is in Figure 3.4(the same codes are also implemented in Tagged view) and that of displayer view is shown in Figure 3.5.



```
@IBOutlet weak var dpWordone: UILabel!
@IBOutlet weak var dpWordtwo: UILabel!
@IBOutlet weak var dpType: UILabel!
@IBOutlet weak var dpNote: UILabel!
@IBOutlet weak var dpImage: UILabel!
@IBOutlet weak var dpCredit: UILabel!
@IBOutlet weak var dpLicense: UILabel!
@IBOutlet weak var imageView: UIImageView!

var wordoneSegue = ""
var wordtwoSegue = ""
var typeSegue = ""
var noteSegue = ""
var imageSegue = ""
var creditSegue = ""
var licenseSegue = ""
var urlSegue = ""

override func viewDidLoad() {
    super.viewDidLoad()

    dpWordone.text = wordoneSegue
    dpWordtwo.text = wordtwoSegue
    dpType.text = typeSegue
    dpNote.text = noteSegue
    dpImage.text = imageSegue
    dpCredit.text = creditSegue
    dpLicense.text = licenseSegue
```

Figure 3.5

In Tagged view, word pairs are classified into groups according to their tags. Word pairs with no tag will be put into the same group called "No tag". Numbers of dynamic cell will change on the base of the number of word pairs having the same tag. The code is shown in Figure 3.6.



```
func numberOfSections(in tableView: UITableView) -> Int {
    return tagNames.count
}

func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    let tagContent = theWordPair.where(tag == tagArray[section])
    all = Array(try! db!.prepare(tagContent))
    return all.count
}

func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String?{
    var tagTitle = ""
    if tagArray[section] == "" {
        tagTitle = "No tag"
    } else {
        tagTitle = tagArray[section]
    }
    return tagTitle
}
```

Figure 3.6

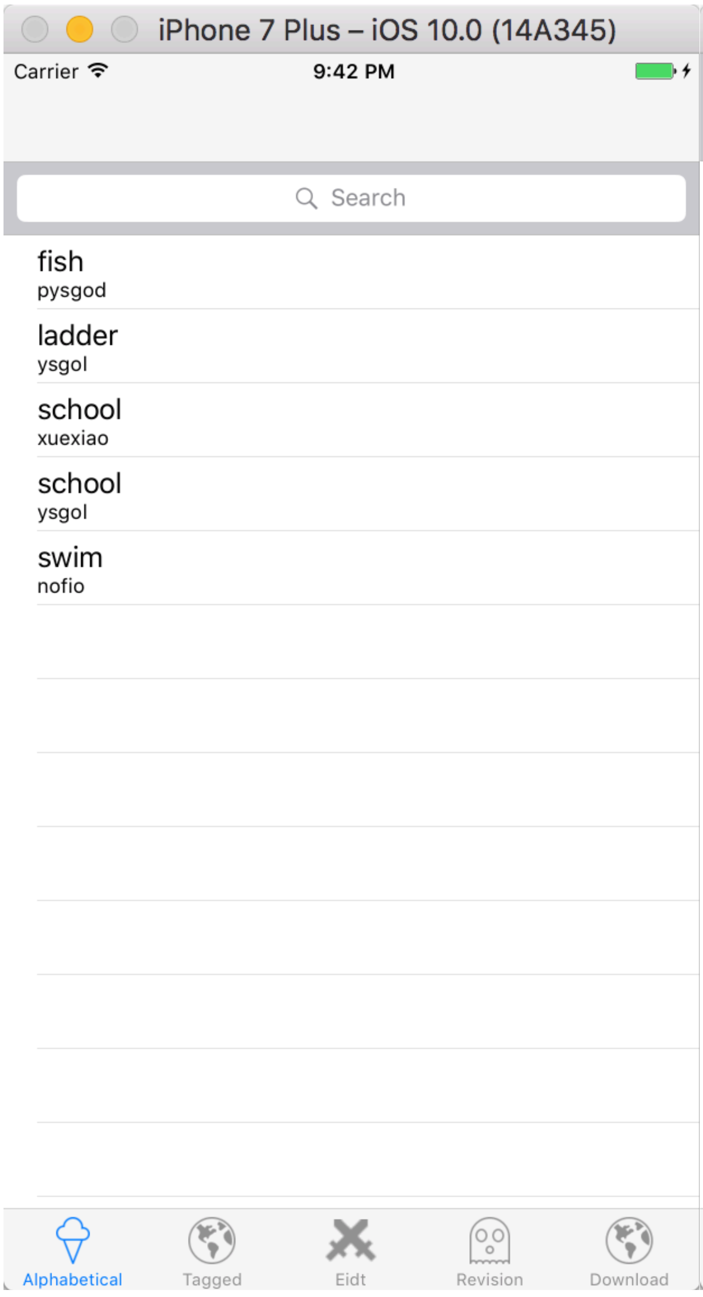
In the displayer function, a third party library, SDWebImage which provides an async image downloader, is implemented for downloading image from a given source URL and displaying the image on the screen.

The implementation of Revision function is simplified greatly for some reasons. It will randomly pick a word pair from database. Two buttons are implemented for displaying wordtwo and next word pair.

In the last part, the download function uses SwiftyJSON to deal with JSON data.

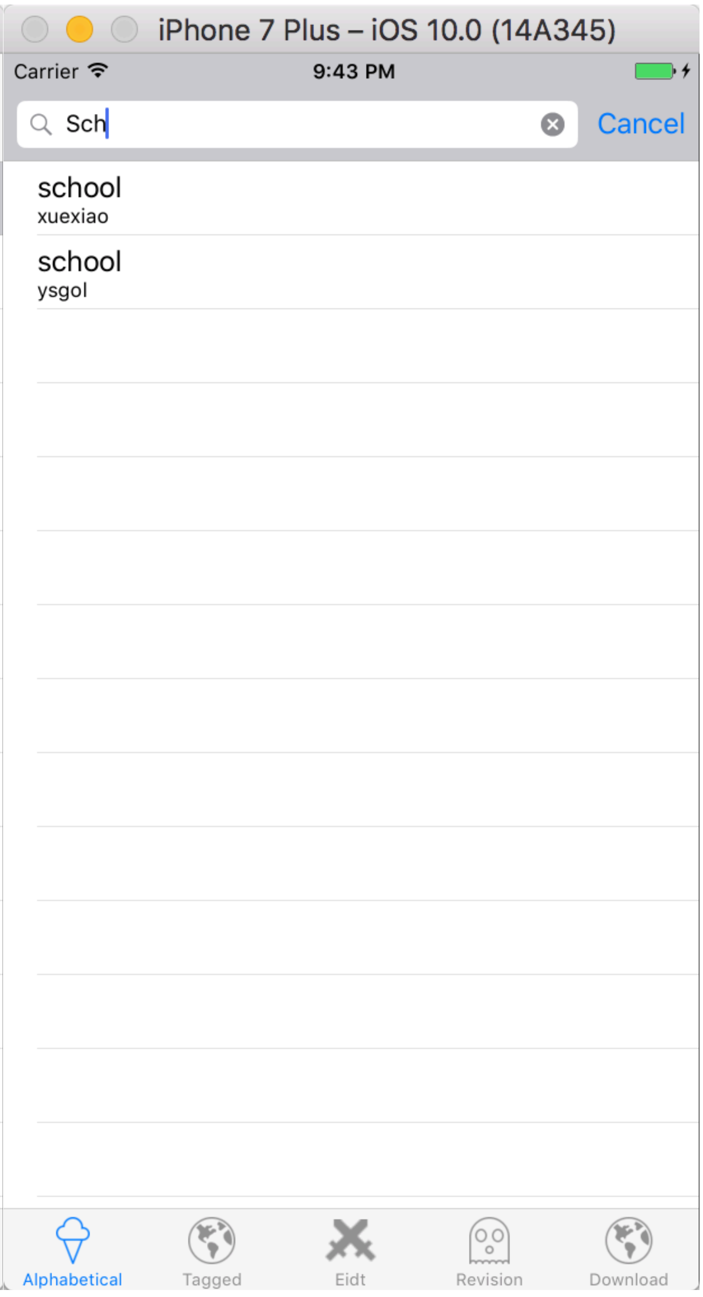
4. Test

- Alphabetical View(Figure 4.1)
- Search Function(Figure 4.2)



fish	pysgod
ladder	ysgol
school	xuexiao
school	ysgol
swim	nofio

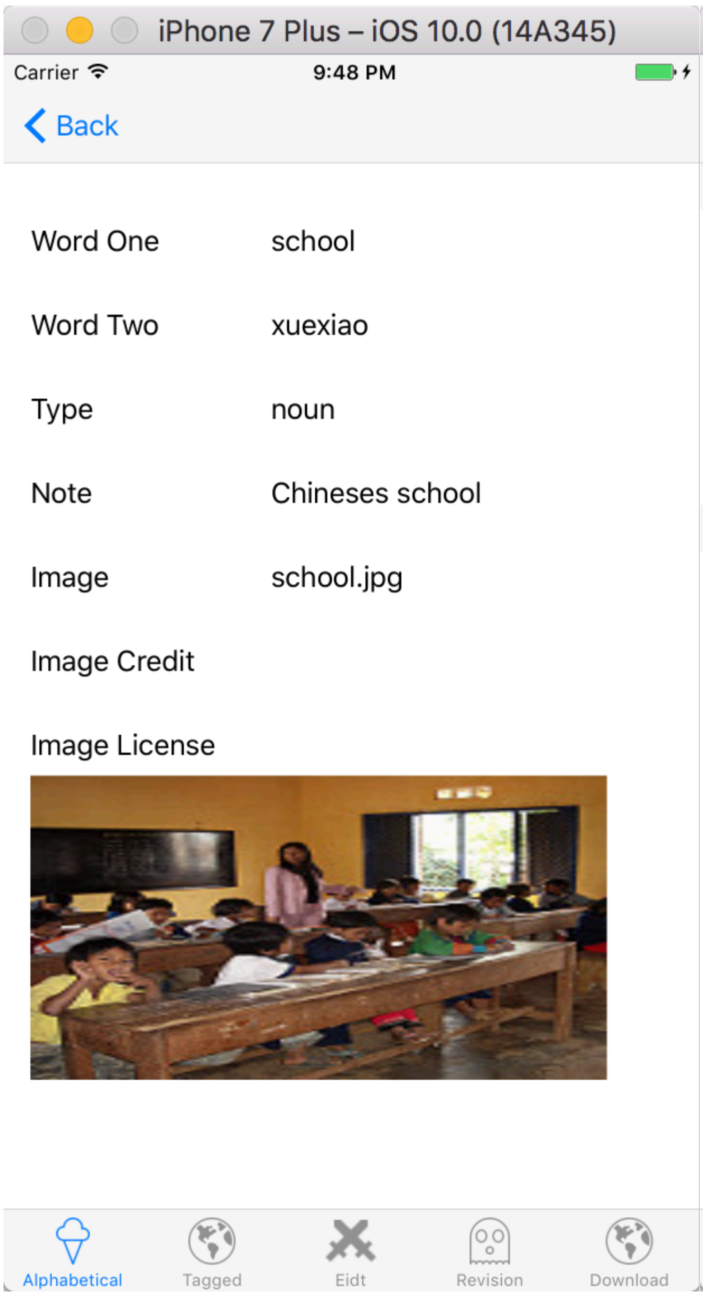
Figure 4.1



school	xuexiao
school	ysgol

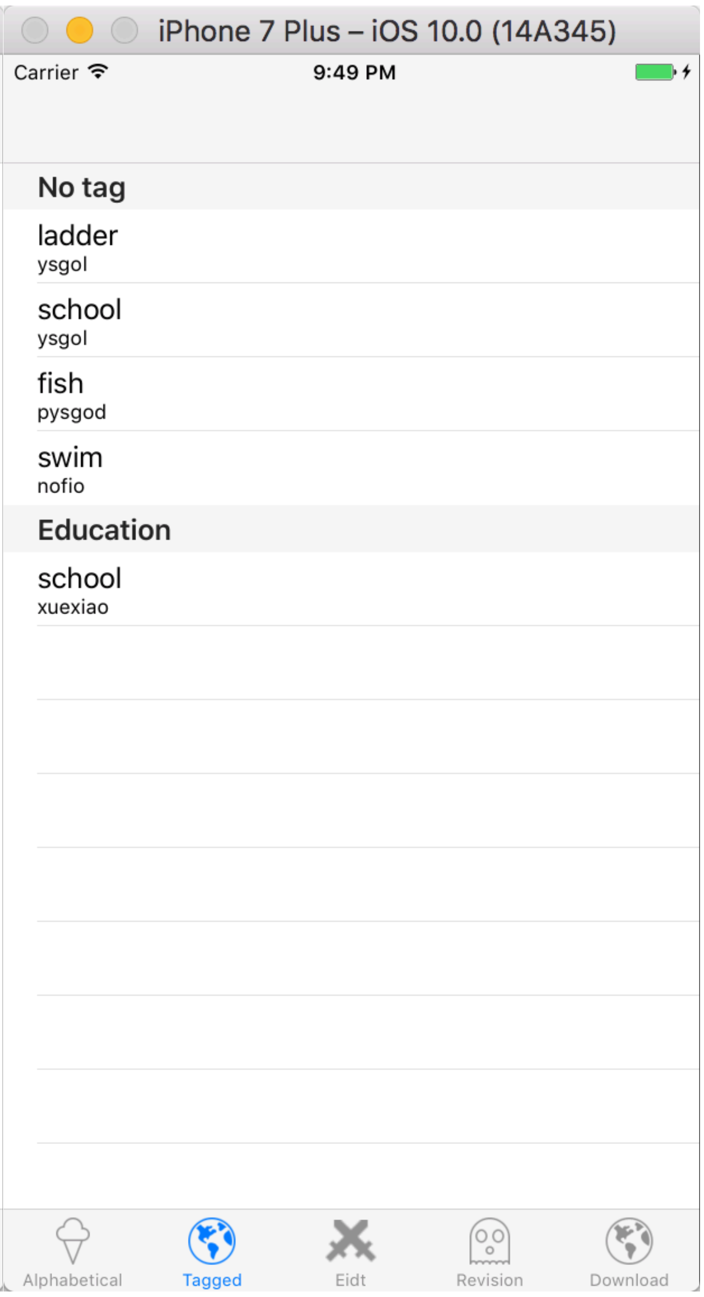
Figure 4.2

- Displayer(Figure 4.3)
- Tagged View(Figure 4.4)



Word One	school
Word Two	xuexiao
Type	noun
Note	Chineses school
Image	school.jpg
Image Credit	
Image License	

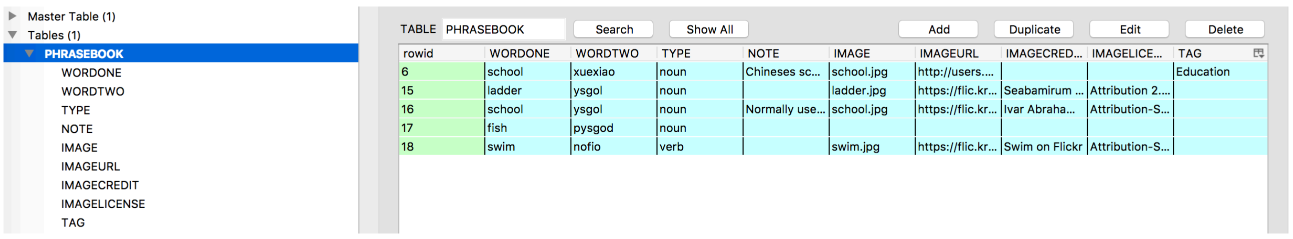
Figure 4.3



No tag	
ladder	ysgol
school	ysgol
fish	pysgod
swim	nofio
Education	
school	xuexiao

Figure 4.4

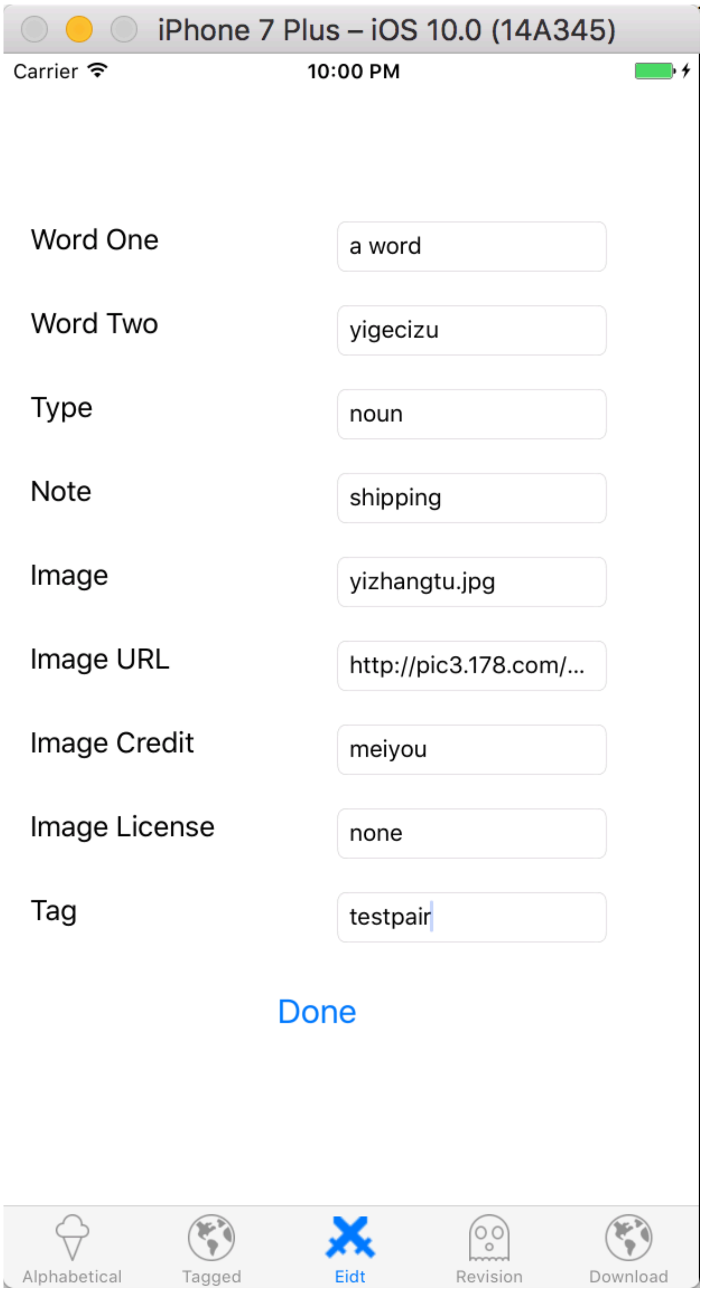
- Database(before adding)(Figure 4.5)



rowid	WORDONE	WORDTWO	TYPE	NOTE	IMAGE	IMAGEURL	IMAGECRED...	IMAGELICE...	TAG
6	school	xuexiao	noun	Chineses sc...	school.jpg	http://users....			Education
15	ladder	ysgol	noun		ladder.jpg	https://flic.kr...	Seabamirum ...	Attribution 2....	
16	school	ysgol	noun	Normally use...	school.jpg	https://flic.kr...	Ivar Abraha...	Attribution-S...	
17	fish	pysgod	noun						
18	swim	nofio	verb		swim.jpg	https://flic.kr...	Swim on Flickr	Attribution-S...	

Figure 4.5

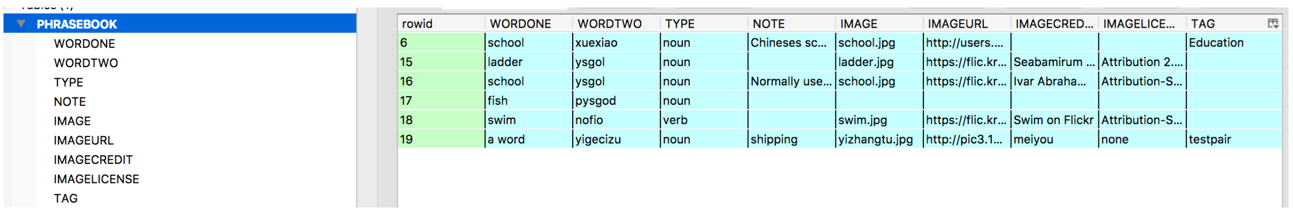
- Edit View(Figure 4.6)



Word One	a word
Word Two	yigecizu
Type	noun
Note	shipping
Image	yizhangtu.jpg
Image URL	http://pic3.178.com/...
Image Credit	meiyou
Image License	none
Tag	testpair

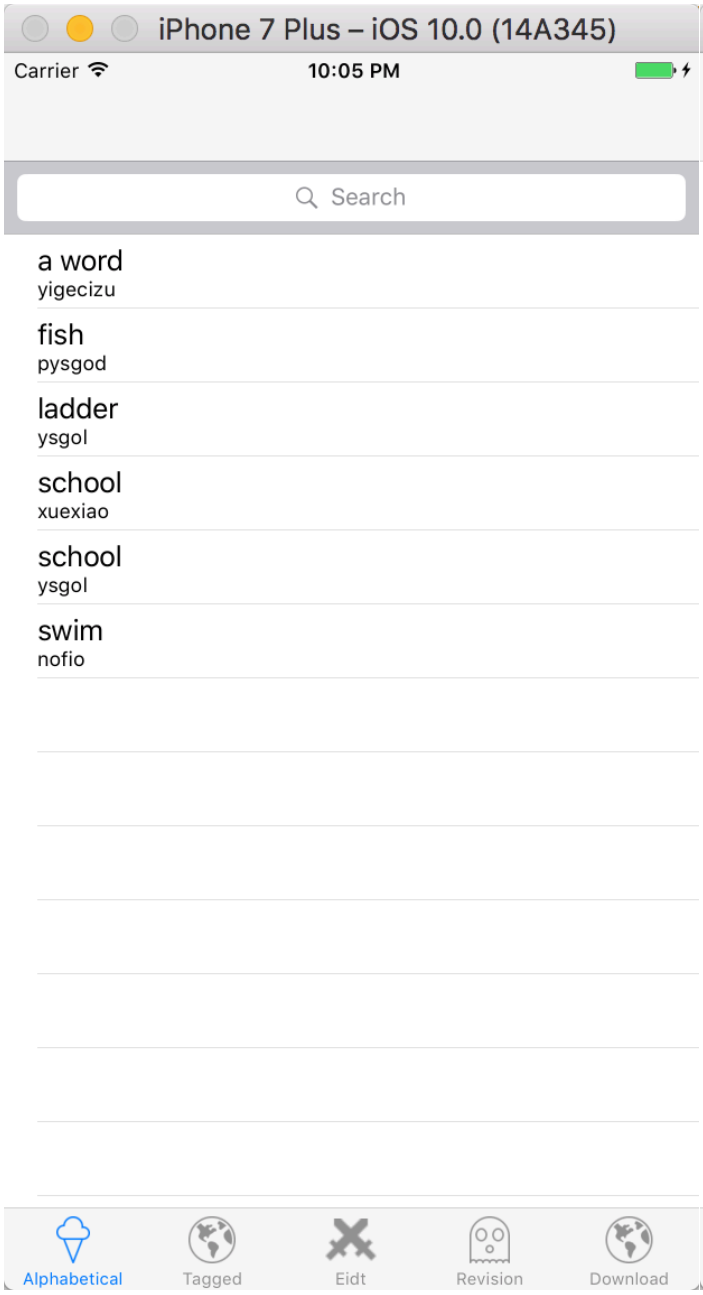
Figure 4.6

- After adding(Figure 4.7, Figure 4.8, Figure 4.9)



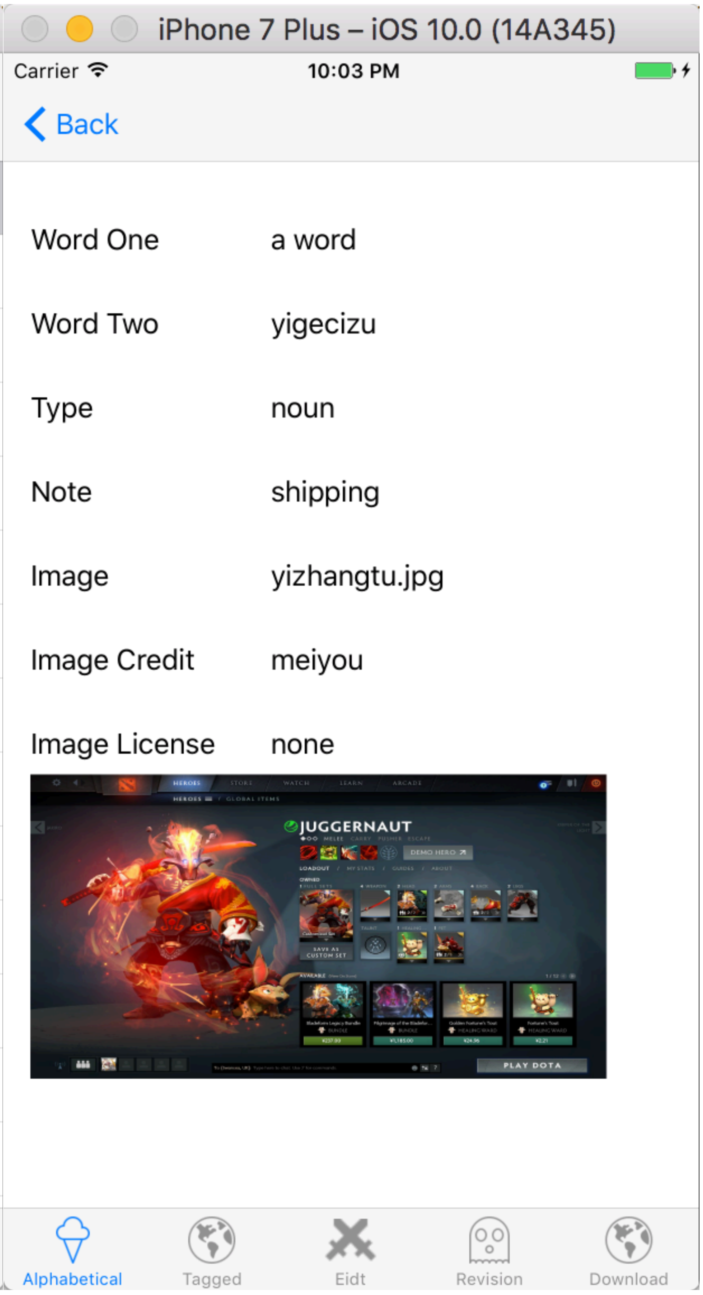
rowid	WORDONE	WORDTWO	TYPE	NOTE	IMAGE	IMAGEURL	IMAGECRED...	IMAGELICE...	TAG
6	school	xuexiao	noun	Chineses sc...	school.jpg	http://users....			Education
15	ladder	ysgol	noun		ladder.jpg	https://flic.kr...	Seabamirum ...	Attribution 2....	
16	school	ysgol	noun	Normally use...	school.jpg	https://flic.kr...	Ivar Abraha...	Attribution-S...	
17	fish	pysgod	noun						
18	swim	nofio	verb		swim.jpg	https://flic.kr...	Swim on Flickr	Attribution-S...	
19	a word	yigecizu	noun	shipping	yizhangtu.jpg	http://pic3.1...	meiyou	none	testpair

Figure 4.7



a word	yigecizu
fish	pysgod
ladder	ysgol
school	xuexiao
school	ysgol
swim	nofio

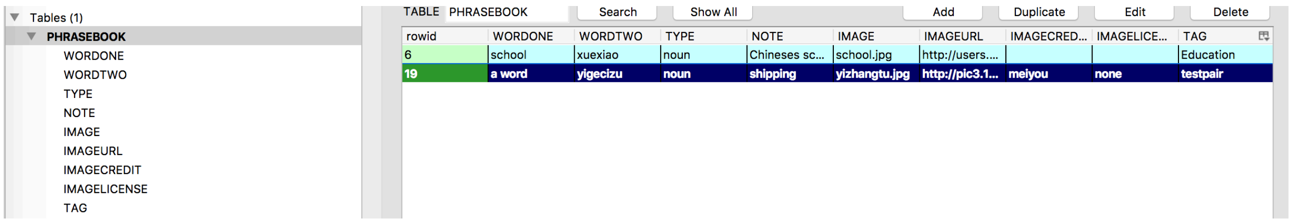
Figure 4.8



Word One	a word
Word Two	yigecizu
Type	noun
Note	shipping
Image	yizhangtu.jpg
Image Credit	meiyou
Image License	none

Figure 4.9

- Download Test(Figure 4.10, Figure 4.11, Figure 4.12, Figure 4.13)



rowid	WORDONE	WORDTWO	TYPE	NOTE	IMAGE	IMAGEURL	IMAGECRED...	IMAGELICE...	TAG
6	school	xuexiao	noun	Chineses sc...	school.jpg	http://users....			Education
19	a word	yigecizu	noun	shipping	yizhangtu.jpg	http://pic3.1...	meiyou	none	testpair

Figure 4.10

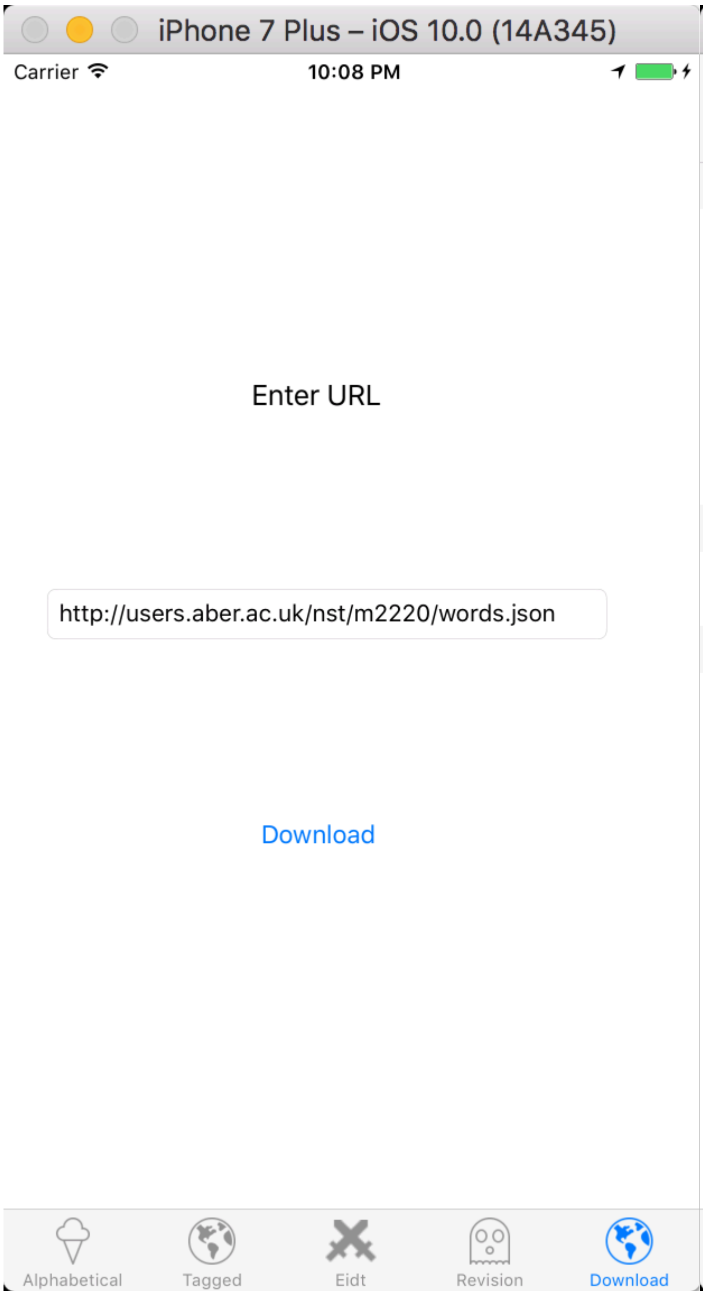
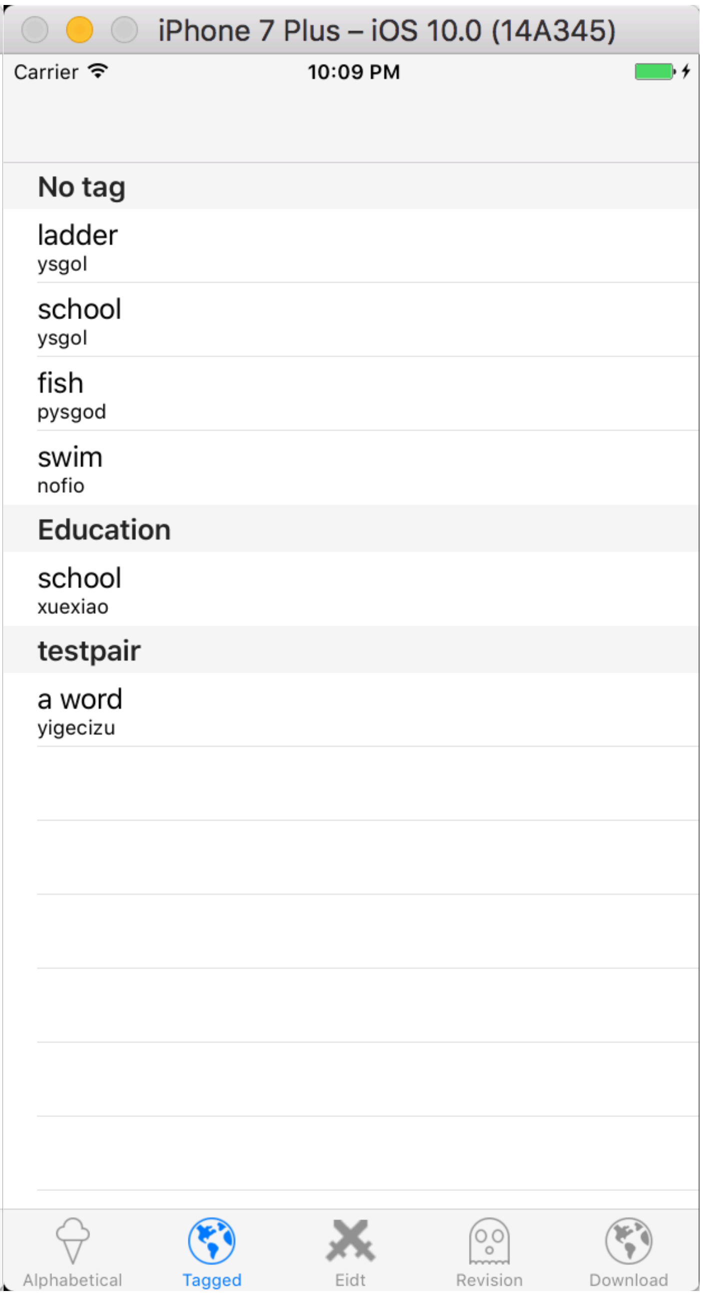

iPhone 7 Plus – iOS 10.0 (14A345)
Carrier
10:08 PM
Enter URL
http://users.aber.ac.uk/nst/m2220/words.json
Download
Alphabetical
Tagged
Eidt
Revision
Download

Figure 4.11



No tag

ladder	ysgol
school	ysgol
fish	pysgod
swim	nofio

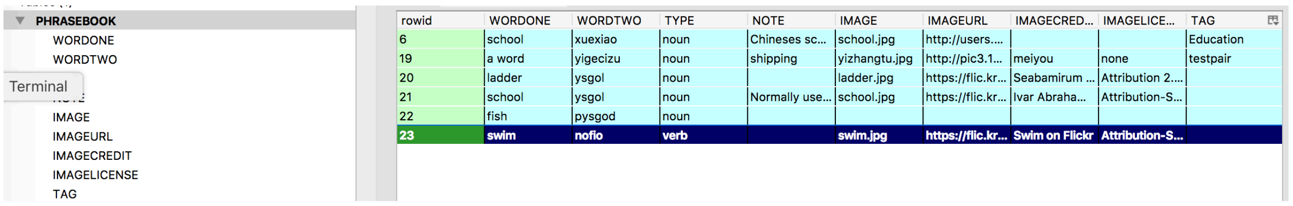
Education

school	xuexiao

testpair

a word	yigecizu

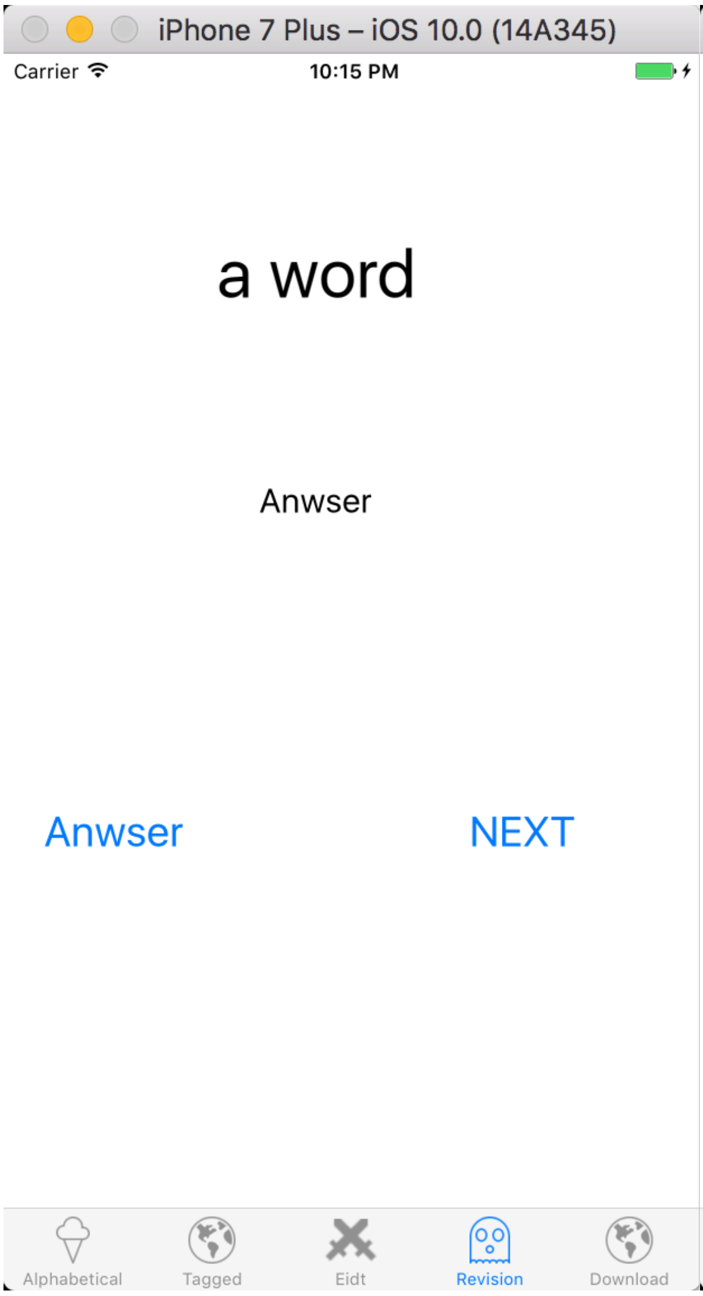
Figure 4.12



rowid	WORDONE	WORDTWO	TYPE	NOTE	IMAGE	IMAGEURL	IMAGECRED...	IMAGELICE...	TAG
6	school	xuexiao	noun	Chineses sc...	school.jpg	http://users....			Education
19	a word	yigecizu	noun	shipping	yizhangtu.jpg	http://pic3.1...	meiyou	none	testpair
20	ladder	ysgol	noun		ladder.jpg	https://flic.kr...	Seabamirum ...	Attribution 2....	
21	school	ysgol	noun	Normally use...	school.jpg	https://flic.kr...	Ivar Abraha...	Attribution-S...	
22	fish	pysgod	noun						
23	swim	nofio	verb		swim.jpg	https://flic.kr...	Swim on Flickr	Attribution-S...	

Figure 4.13

- Revision Test(Figure 4.14, Figure 4.15)


iPhone 7 Plus – iOS 10.0 (14A345)

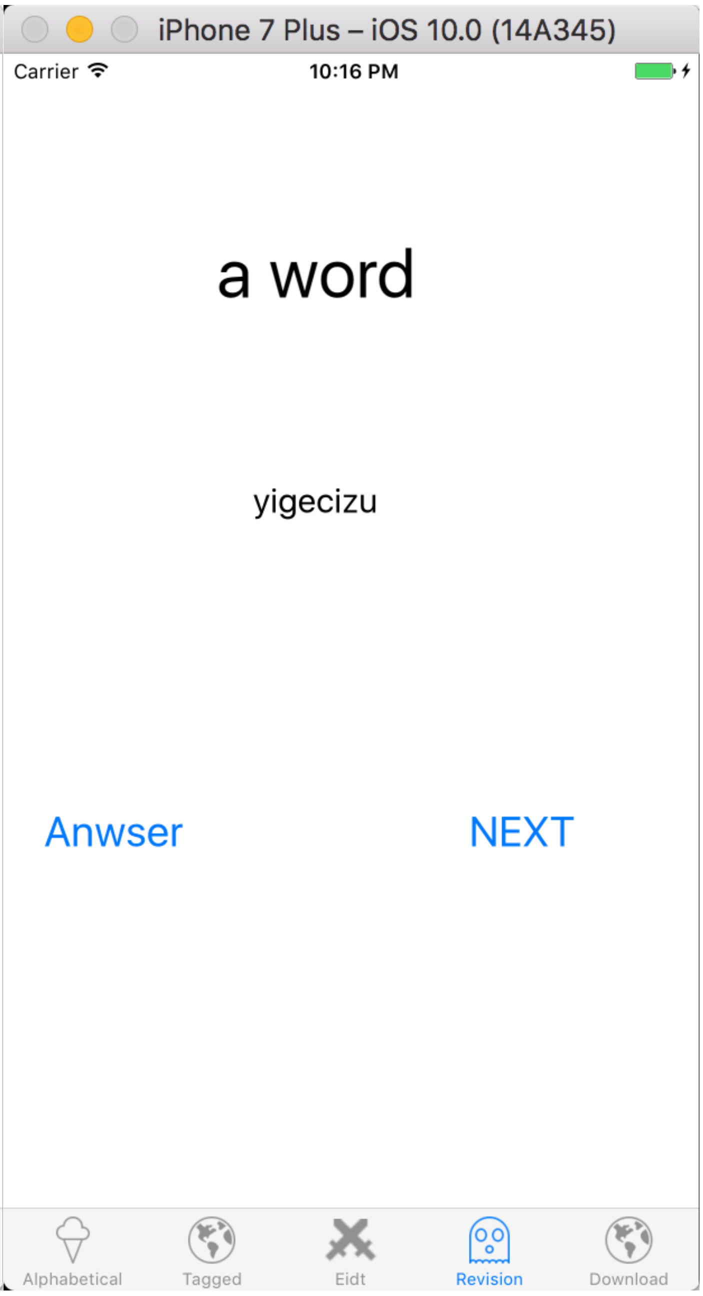
Carrier 10:15 PM

a word

Anwser

Anwser NEXT

Figure 4.14


iPhone 7 Plus – iOS 10.0 (14A345)

Carrier 10:16 PM

a word

yigecizu

Anwser NEXT

Figure 4.15

5. Conclusion

Firstly, the application runs well currently and no bug has been found yet. Though it does not full fill the requests, it can still take basic jobs of a phrase book.

Secondly, in this project, knowledge of developing an iOS application is increased a lot. We learned how to develop such an application through Xcode and also how to use third party libraries to help developments.

Adding multiple tags for a word pair has not been figured out so far. However, similar functions have been implemented in some shopping apps which allow users to filter some attributes of goods.

To implement the revision function, we could give every word pair an additional attribute to record the accuracy and update it with the average of former accuracy and the current accuracy(0% for wrong answer and 100% for correct). This can make recent revision has more impact on the accuracy rate. Then, there could be an operator function to weight these probabilities and give these wordpairs different possibility to be chosen. The later part could be quite difficult to implement, but such a method could help build an appropriate revision function.