

# iOS Welsh Vocabulary App

*SEM2220 Assignment 2*

ALEXANDER D BROWN (ADB9)

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Development</b>	<b>3</b>
2.1	Implementation of Feature Requests . . . . .	3
2.1.1	FR1: Adding Fields to the Word Link . . . . .	3
2.1.2	FR2: Update the Custom Table Cell . . . . .	4
2.1.3	FR3: Add Word Link Detail Screen . . . . .	4
2.1.4	FR4: Create Word Games . . . . .	4
2.2	Problems Encountered . . . . .	5
<b>3</b>	<b>Testing</b>	<b>5</b>
<b>4</b>	<b>Evaluation</b>	<b>5</b>

# 1 Introduction

This report shows the process undertaken to make the pre-existing iOS Welsh Vocabulary Application to meet the following features requests:

1. Adding fields to the word link.
2. Updating the custom cell.
3. Adding word link detail screen.
4. Revision game.

The existing application provides the basic functionality to display words from a SQLite database as well as the loading and saving of a single translation with no context, area or notes to the database.

## 2 Development

This section describes the process taken to implement the four feature requests required of the application, as well as detailing the problems encountered and how they were solved.

Finally this section also discusses the extra features implemented to improve the user experience.

### 2.1 Implementation of Feature Requests

This section describes how each of the four feature requests were added to the existing application.

#### 2.1.1 FR1: Adding Fields to the Word Link

The first task was to allow users to add the following extra details:

- Context
- Area
- Note

All these fields were added as properties of the WordPair object.

The context is a simple, single line string, which, like the English and Welsh fields, could be added as a single text field in the UI and required altering the Word Link table to add a context column with type text.

The area can be one of three items:

- North
- South
- Both

The iOS segment selector was the perfect tool for displaying this information in the UI (with the default of Both).

An enumeration was created in code to store this information, with methods to translate from a String or character to the enumeration construct and vice versa.

In the database this information was represented as text for simplicity, it could also have been represented as a character if optimisation was an issue.

Additional methods in the `WordPair` class to translate numbers (which the segment control produces) to the enumeration type.

Finally user notes were added as a text input field and were added to the database as a text field.

### **2.1.2 FR2: Update the Custom Table Cell**

The next feature was to update the table cell to display Word Link information. I went with a very simple representation, only displaying the word and all its possible translations.

### **2.1.3 FR3: Add Word Link Detail Screen**

The next requirement was to show the details of a word link on a next screen.

To show this information a second table was design, showing each translation with the context of that translation.

The user can then click on these cells to show the same information that was given to add words, using a screen laid out similarly to the add words screen.

### **2.1.4 FR4: Create Word Games**

Two word games were implemented, one was a simple game to select the translation from a list which contained the correct translation.

The other game is based on the popular game hangman, but gives the translated word as a clue (but trades this off with fewer guesses).

#### **Translation Word Game**

The translation game requires users to pick the correct translation from four possible words. This process happens ten times and a score is shown at the end.

Rather than load the ten words before the game starts, it seemed easier to select the four word pairs from the database and then randomly select the correct word to display. This does allow duplication of questions, but allows a database size of anything greater than four to be used.

To display the four choices, a `UIPickerView` was implemented and the `TranslationGameViewController` was also made a delegate for this picker. To make this process easier, the correct position of the word in the array the picker uses is stored and then used to check if the choice is correct.

Initially, the idea of segueing back to the same view was considered, but it was found that simply manipulating the view dynamically was much easier to deal with.

To segue at the end of the game two different buttons are used. The first is visible through the entire game until the last question, at which point it becomes hidden and another, previously hidden, button is made visible to perform the segue to the results page.

#### **Hangman Game**

The hangman game was marginally easier to implement, but did bring about some more radical changes to existing code.

Previously, the translation game had handled the random selection of words but, with another class which needed to perform the exact same logic, this code was moved into the `SharedData` class. This keeps the concerns of loading data in a single place.

It also affected the `WordPair` class to have better methods for providing words in different languages. Previously a few helper methods had been implemented to get a word from a certain language and the translation from a certain language, required because both games would only know the language the answer was in.

This mean that guessing the Welsh word also involved guessing the context (including brackets and spaces). This was changed to pass in a boolean requesting the context.

The images were displayed in a `UIImageView` and were loaded into an array when the view loads. A call to the methods to set the image for the view was then used with the index of the image to display which corresponded to the number of lives the user had left.

## 2.2 Problems Encountered

Some of the main problems encountered in this project were due to the lack of familiarisation with Objective C and the iOS framework. Sites like the iOS documentation and StackOverflow were used to gain knowledge of what should be done to solve a problem and how to debug common problems.

One problem that was found was more of a C issue than an Objective C problem; circular dependencies.

The initial code has the language setting enumeration in the `SharedData` class, which includes the `WordLink` header which, in turn, includes the `WordPair` header.

Later in the project the language settings were needed in the `WordLink` class to provide extra functionality required for the feature requests. At this point the `WordLink` header also included the `SharedData` header. This made the enumeration inaccessible in the `WordLink` header despite the fact it would seem it should be.

Initially a C style fix was employed using macros, but this was quickly remedied by moving the enumeration into the `WordLink` header, and later into the `WordPair` header for similar reasons.

## 3 Testing

This section describes the testing performed on the application, specifically using the iOS simulator.

## 4 Evaluation

Part	Worth	Predicted Grade
Documentation	30%	25%
Implementation	50%	45%
Flair	10%	5%
Testing	10%	7%
Total	100%	82%

Table 1: Break Down of Marks