Software Engineering Group Project

Welsh Vocabulary Tutor - Project Maintenance Manual

|  |  |
| --- | --- |
| Author: | Sophie Janssens [soj11] |
| Config Ref: | SE.MM.09 |
| Date: | 4th May 2020 |
| Version: | 1.0 |
| Status: | Draft |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2019

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. PRogram description 3

3. Program Structure 3

3.1 Modules 3

4. Algorithms 5

5. Main data Areas 5

6. Files 5

7. Interfaces 6

8. Suggestions for improvement 6

9. Things to watch for when making changes 6

10. Physical limitations of the program 7

11. Rebuilding and testing 7

REFERENCES 7

DOCUMENT HISTORY 8

# Introduction

## Purpose of this Document

The purpose of this document is to answer all the likely questions that an installer or maintainer may have regarding our program source.

## Scope

This covers all of the possible questions an installer or maintainer may have regarding our programs purpose, structure and contents. It also explains how to set up the program, any rules that may need to be observed, improvements and pitfalls for future maintainers looking to adapt and develop it, and the physical limitations of the program. It also explains what to do if any issues are found.

Anyone installing or maintaining this software should read this document, this includes the client and all project members.

## Objectives

The main objective it to aid any future maintainers in developing the program further, making them aware of any existing bugs or thing to watch out for, it also suggests improvements for the program that they may wish to make. Its other objective is to help installers by describing the physical limitations of the program and describing how to set up the program and rebuild it.

# PRogram description

This is a welsh vocabulary tutor. It allows a welsh learner to look up the meaning of new words and practice them until familiar with them and they feel they have become part of their vocabulary.

It initially provides a very basic list of welsh vocabulary; this is referred to as the dictionary. Learners can the select words from the dictionary that they wish to learn, these words go into a practice list which they can then quiz and test themselves on. Due to people wanting to learn the language for different purposes and having different interests the user may also add and remove words from the dictionary.

This program is written in Java and uses JavaFX and JSON (Java Simple Object Notation) to meet the functionality of this program. It uses linked lists of word objects taken from a JSON file to represent the dictionary. It then searches words using a linear method and sorts the words in the dictionary using two different comparators for English and welsh. It makes use of classes and JavaFX to allow the user to practice words.

# Program Structure

This program is split up into several modules, each module grouping together files and classes by purpose. Any classes outside these modules, *Main*, *Settings*, *UIHandler*, are used to display and run the program and do not fit into a cluster of classes.

## Modules

### assets

This contains two PNG files that contain images, one of an UK flag, the other of the Welsh flag. These files are used in the UI handler class to set the graphics for the language button. The language button refers to what language the list is currently sorted by.

uk.png – contains image of UK flag.

welsh.png – contains image of the Welsh flag.

### css

This contains all the css (Cascading Style Sheets) files in the program, these are responsible for the sizing of the font for the visual output of the program. There are 3 files set - this is due to *Settings* containing a function that allows the user to select one of the three font sizes, the files then being applied to the table in the *UIHandler* class.

big.css – this contains the value for the largest font size, 24px.

medium.css – this contains the value for the medium font size setting, 18px.

small.css – this holds the value for the small font size, 12.

### dictionary

This package contains the classes *Dictionary* and *PracticeList*, this deals with loading the lists from the json file and setting up the dictionary and practice list, so they may be used and implemented in the rest of the program. It contains all the gets and sets and methods for both lists.

### quizzes

This module deals with all 4 of the test methods specified in the requirements [2]. It contains a class for each type of test, *FindWordDefinitionQuiz*, *MatchFourQuiz*, *WriteDefinitionQuiz*, and flash cards, *FlashCard*, as well as a class which manages the quiz, *Quiz*. The quiz is a collection of all the methods of testing and so keeps track of the score and progress of the user. All these classes have been grouped into this module due to them dealing with the revision of the practice list. They are then implemented in the *UIHandler* class.

### scenes

*scenes* contain all the fxml files for the project. Each fxml depicting the layout and objects for a scene in the program.

addWordScene – formatting for the page that allows you to add a word to the dictionary, lets the user input information so the word may be added.

flashcardScene – formats the page which allows the user to view the flash cards, all of which are words from the practice list.

homeScene – formats the page that the user sees when they first run the program, displays the dictionary.

letterSelectScene – formats the page which allows them to select a letter which will then narrow down the dictionary list they are looking at.

messageBoxScene – this is used to format stand-alone message boxes and simply contains single button and label.

quiz1Scene – this establishes the core components needed for the quiz in which the user must select the correct definition for a word out of 6 given

quiz2Scene – this establishes the core components needed for the quiz in which the user must select type in the definition for a given word

quiz3Scene - establishes the core components needed for the quiz in which the user must match 4 English words to 4 Welsh definitions

revisonListScene – this formats the Practice List page where the viewer and see the table containing their practice list and may interact with its elements.

revisionScene- this formats the page which allows the user to choice what type of revision they wish to perform, Flashcard or Quiz.

settingsScene – this formats all the components for the page that allows the user to select their personal settings.

### tests

This contains the Junit module tests, *TestDictionaryFunctionality*, *TestPracticeList*, these check these two important classes match the functionality specified in the requirements of the program[1]. *TestDictionaryFunctionality* contains the two test methods, *testAddWord* which tests if a word can be successfully added to the dictionary and if it is also checks that it is added to the practice list. It also contains *testDictionaryMaintenance* which checks that any new words added to the dictionary are saved and still present when the dictionary and practice list are reloaded.

*TestPracticeList* the two test methods *maintainingPractice* and *removeWordFromPractice*.

*maintainingPractice* tests that when a new word is added to the practice list it is saved and is still present when the practice list is reloaded.

*removeWordFromPractice* tests that when a word is added to the practice list it can then be successfully removed from the list.

### word

This module contains classes, *EnglishComaprator*, *WelshComparator* and *Word*. The purpose of this is to group together all the classes that deal with representing and sorting the vocabulary.

A description of all of the methods in each module can be found in document Welsh Vocabulary Tutor Design Specification Standards[1], section 4. A set of diagrams that showing how the classes interact and which methods call which are the sequence diagrams, there is one for each of the most significant actions in this program. These can be seen in section 5.1 of our Welsh Vocabulary Tutor Design Specification Standards[1].

# Algorithms

We have several significant algorithms used in the program most of these are used in the *Quiz* as this is where we found the most complex algorithms to be. These algorithms allow a randomly ordered list of words from the practice list to be used in the quiz and allow a random order of different types of tests in the quiz. A detailed description of these significant algorithms and how they work can be found in section 5.2 of our Welsh Vocabulary Tutor Design Specification Standards[1].

# Main data Areas

The significant data structures that store the important information in our program are discussed in section 5.3 of our Welsh Vocabulary Tutor Design Specification Standards[1].

# Files

The program assumes it has three JSON files that contain all the personalised data for the user. These fixed files are in the current directory src where the main package uk.ac.aber.cs221.nine is contained.

dictionary.json – this stores all the words that are in the users Dictionary, this should include their own personal words they have added

practiceList.json - this stores all the words that are in the users Practice List, this should include the words they last had in their own personal list.

settings.json – this stores all the data for the settings so next time the program is run their last used settings haven’t changed.

There are also two png files contained in the aforementioned assets package that allow the use of icons in the program.

# Interfaces

There is not much to the protocols for this program there are however a few things required for it to run due to the use of json, JavaFX and JUnit files. Upon opening the project, the *javafx-sdk-13.0.1* library needs to be added this can be done by going to

File -> Settings -> Appearance and behaviour -> Path Variable

Here the library must be added, and the variable named PATH\_TO\_FX, the value of the path is where your javafx-sdk has been stored. Then Apply to add this.

The run configurations must also be changed, this is done by going to

Run -> Edit Configurations -> Application -> Main>

In the textbox labelled

<VM options>

Add the line below

--module-path ${PATH\_TO\_FX} --add-modules javafx.controls,javafx.fxml

Then Apply the changes.

If the JavaFX library also needs to be added to the project structure, to do this go to

File -> Project Structure -> Project Settings -> Libraries

Add a new project library and select the *lib* file from the **javafx-sdk-13.0.1** and APPLY the changes.

The junit libraries must also be added, this is done simply by going to

File -> Project Structure -> Project Settings -> Libraries

Adding a new project library and selecting JUnit 5.4, the junit files from your computer.

To add the json-simple libraries the same steps must be done as with the javafx libraries but with json-simple-1.1.1 file.

# Suggestions for improvement

Due to time constraints there are still things we wish to improve upon in our program. One of these improvements that has been omitted but that could be made in the future is the addition of a super class to the dictionary and practice list. They have a lot of similar functionalities and methods as well as both classes making use of the same type of data structure. This was considered when designing this program but was not implemented, however may be helpful if the program was developed in that way further.

To make the system more robust, an enum could have been implemented. This could have been used to define the three set word types, reducing the possibility of someone breaking the code if they wished to continue developing it later. It would prevent invalid values being used and help with consistency.

To make the code more adaptable and reusable the UI aspect of the program could be further segregated from the functionality and backend development of the project. The UI could then be more refined, and the functionality of the code could be more adjustable for future use and changes.

The structure of the algorithms used could also be improved if given more time, presently the program makes use of essentially a linear search. If the list of words being searched became greater this could have a noticeable effect of the efficiency of the system, a better search algorithm may then be of use.

# Things to watch for when making changes

When making changes to the project we are not aware of many things that may cause a knock-on effect. The type of file that the dictionary is being loaded from may pose an issue. If changing the file that you wish to load the dictionary, settings or practice list from, any file that isn’t .json would cause an issue due the program using json objects and other objects and functions specific to the file type.

If a new word type was to be added to the program this may pose an issue as changes would have to be made through out the program, this can be made easier though by using an enum as discussed in the section above.

# Physical limitations of the program

There so far seem to be no extreme physical limitations to this program, it runs on a computer with 8.00 GB of RAM a CPU of 2.70GHz and a disk space of 256 GB. It should run on most home computers.

# Rebuilding and testing

All of the program files can be found in the *src* folder on our git repository, any documentation of the project and program can be found in our *docs* folder. When rebuilding the program almost all of the files are included, the json, juint and javafx files can however be found online below are links to when you can find them.

javafx- <https://openjfx.io/>

json- <https://code.google.com/archive/p/json-simple/downloads>

juint- <https://junit.org/junit5/>

To rebuild the system, follow the rules in section 7, Interfaces, this tells you how to implement the files mentioned above and should instruct you on how to rebuild the program.

The module tests for the program are contained in the test package in the program and are run by selecting them, right clicking and selecting run. First the juint libraries must be installed. A notification will be displayed telling you if the which tests have been successful. Any other tests are described in the Welsh Vocabulary Tutor Test Specification [3] and the results of these tests can be found in our test report[4].

If a new issue is discovered in the program a test should be added to the Welsh Vocabulary Tutor Test Specification [3], where the following information for the test must be recorded:

Test reference – the next sequential number in that testing section

Requirement being tested – the list of requirements can be found in the Requirements Specification [2], or in the User Interface Standards document[5]

Test Content – a description of what the test is for, what it does

Input – the input into the program that is required for the test

Output – what the program should output

Pass Criteria – the criteria required for a test to be considered as passing

REFERENCES

All of the below can be found on our git repository.

[1] SE.DS.09 - Welsh Vocabulary Tutor Design Specification Standards

[2] QA Document SE.QA.CSRS - Welsh Vocabulary Tutor Requirements Specification

[3] SE.TS.09 - Welsh Vocabulary Tutor Test Specification

[4] SE.TR,09 – Welsh Vocabulary Tutor Test Report

[5] SE.UIUC.09 - Welsh Vocabulary Tutor User Interface Use Case Document

DOCUMENT HISTORY

| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 1.0 | N/A | 05/05/20 | N/A - original version | soj11 |