Software Engineering Group Project 09

Welsh Vocabulary Tutor Design Specification Standards

|  |  |
| --- | --- |
| Author: | Dixon Chu[dkc2], Kassy[kab71] |
| Config Ref: | SE.DD.09 |
| Date: | 6th May 2020 |
| Version: | 2.3 |
| Status: | Release |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2019

**CONTENTS**

[1. INTRODUCTION 3](#_Toc39644915)

[1.1 Purpose of this document 3](#_Toc39644916)

[1.2 Scope 3](#_Toc39644917)

[1.3 Objectives 3](#_Toc39644918)

[2 Decomposition description 4](#_Toc39644919)

[2.1 Program description 4](#_Toc39644920)

[2.2 Significant classes in the program 4](#_Toc39644921)

[2.3 Mapping from requirement to classes 4](#_Toc39644922)

[3 Dependency description 5](#_Toc39644923)

[3.1 UML diagram 5](#_Toc39644924)

[3.3 Component diagram 5](#_Toc39644925)

[4 Interface description 6](#_Toc39644926)

[5 Detailed design 14](#_Toc39644927)

[5.1 Significant algorithms 14](#_Toc39644928)

[5.2 Significant data structure 14](#_Toc39644929)

[5 .3 Sequence diagram 14](#_Toc39644930)

[ Flip flashcard get opposite language 22](#_Toc39644931)

INTRODUCTION

## **1.1 Purpose of this document**

The purpose of this document is to give a description of the design elements necessary for the implementation phase. This document is the version of the design for the CS22120 Welsh Tutor application project.

## **1.2 Scope**

This document specifies the architecture and design for the implementation of the Welsh Tutor application. It describes the necessary layout and content of the design, the mapping from the requirement to classes, an outline specification for each class, a UML diagram, a component diagram, a sequence diagram and an object diagram.

It is intended for all the team members, managers and the client.

1.3 Objectives

The main objective is to aid the production of a design specification which is a complete and accurate translation of the client’s requirements into a description of the design elements necessary for the implementation phase. These will generally include the software structure, components, interfaces, and data. In a complete design specification, each requirement must be traceable to one or more design entities.

1. DECOMPOSITION DESCRIPTION
   1. Program description

The program consists of opening an application which directs us to a dictionary sorted by English or Welsh according to the choice of the user, it also accommodates a favouritism system to facilitate the learning of the user. Each word marked as a favourite is sent to the Practice list, then the program allows the user to "Practice", this is where the user can choose two options to learn these words. Flashcard, which shows a word then the user can choose to show the translation when they want, or Quiz which takes the words from the practice list and uses them in different tests.

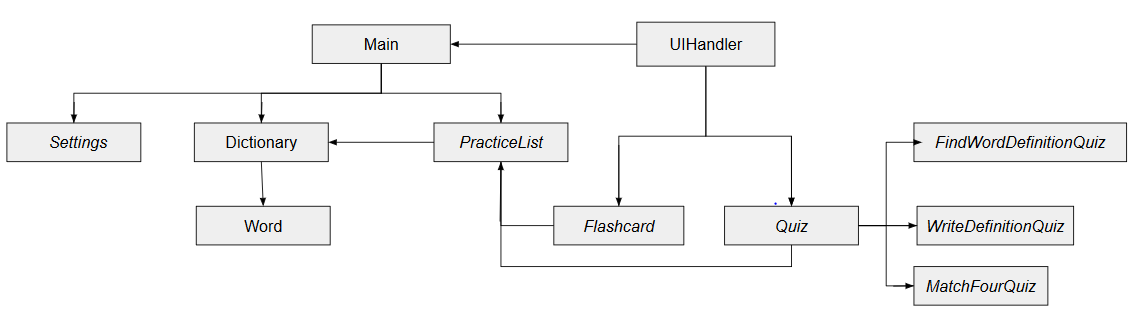
2.2 Significant classes in the program

* **Main** - The Main class holds everything together and it is where the main methods run that initialise the main data structures and start the application.
* **Dictionary** - The Dictionary class that stores and manipulates a list of Words representing the dictionary, it contains a collection of words English/Welsh/Wordtype in a linked list. From this class the words my be added removed and loaded from the json file provided.
* **PracticeList** - The Practice List class contains a list of words chosen by the user from the dictionary. From this class they may be manipulated to be personalised by the user and to fulfil the functionality of the quizzing requirement.
* **Word** - The Word class contains all the English, Welsh and Word Type objects. Representing an element in the dictionary.
* **UIHandler** -This class handles all the possible JavaFX scenes, dealing with the communication between the classes and these. This allows for user interaction, and the sorting of the dictionary and practice list as well as many other important functionalities.
* **FlashCard** - The FlashCard class retrieves the words from the practice list and shows either English or Welsh version of the word one by one Flashcard.
* **Quiz** - The Quiz class has various types of tests for the user to learn the words from their practice list. Each test takes the words from the practice list and the user must choose the correct answer among a random list of words taken from the dictionary and one of them will be the correct translation. This class makes up a list of words taken from the practice list and shuffles them, this list of words is then which each test can be based around a word from the list.
* **FindWordDefinitionQuiz** - The FindWordDefinitionQuiz class gives a word in one language and a list of 6 possible meanings in the other language, so that the user can select the correct translation.
* **MatchingFourQuiz** - The MatchingFourQuiz class gives four words in one language from the practice list and four translations of these words for the user to match.
* **WriteDefinitionQuiz** - The WriteDefinitionQuiz class,the user is given a word and types in the translation.
  1. Mapping from requirement to classes

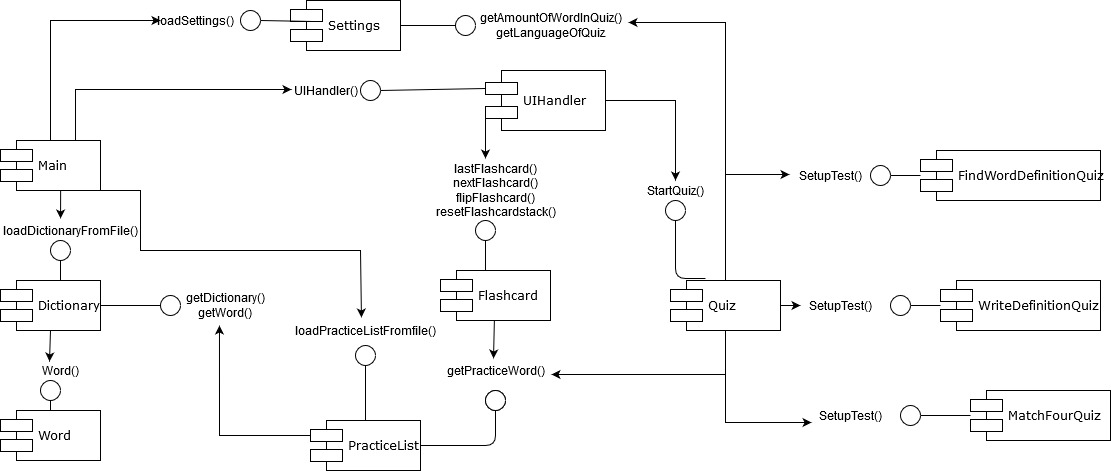
|  |  |
| --- | --- |
| *Requirement* | *Classes providing requirement* |
| FR1 Startup | UIHandler, Dictionary, Word |
| FR2 Ordering of the list | UIHandler |
| FR3 Searching of list | UIHandler, Dictionary, PracticeList, Word |
| FR4 Maintaining a practice list | PracticeList, Dictionary, UIHandler |
| FR5 Adding new words to the dictionary | UIHandler, Dictionary, Word |
| FR6 Display of words | UIHandler, Dictionary |
| FR7 Reviewing the practice list | UIHandler, PracticeList |
| FR8 Flashcards | UIHandler, PracticeList, Flashcard |
| FR9 Tests on practice words | UIHandler, PracticeList, Quiz |
| FR10 Running tests | UIHandler, PracticeList, Quiz |

1. DEPENDENCY DESCRIPTION

3.1 UML diagram

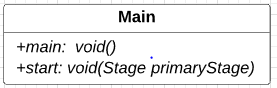


3.2 Component diagram

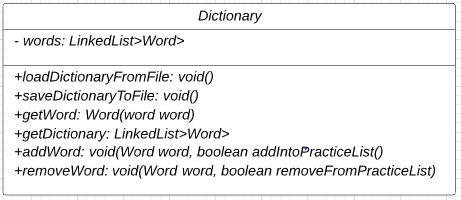


INTERFACE DESCRIPTION

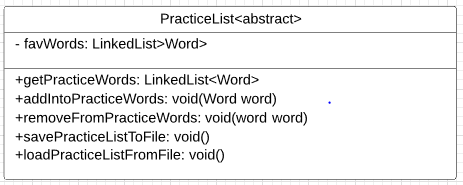
* **Main -** Holds all the classes together and has the main method to run the application.



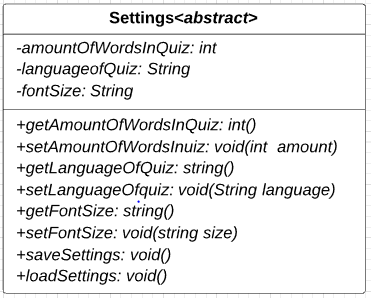
* + main : Where the program run.
  + start: Loads Settings, Dictionaryand PracticeList from json files, creates an instance of UIHandler, saves Settings, Dictionaryand PracticeList to json files when the Application closes and initialises the UI.
* **Dictionary -** Contains a collection of words English/Welsh/wordtype.



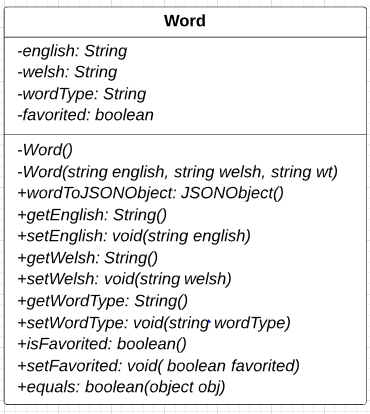
* + loadDictionaryFromfiles: Loads the Dictionary JSON File
  + saveDictionaryToFile: Saves new words added to Dictionary JSON File.
  + getWord: Search a word in Dictionary, if the word is there then return the word.
  + getDictionary: Get the list of word from Dictionary.
  + addWord: Add new word into the Dictionary list.
  + removeWord: Remove a word from Dictionary and if this word is in the PracticeList then remove it from there too.
* **PracticeList -** List of the words chosen by the user from the dictionary to revise, with an option to add and remove them.



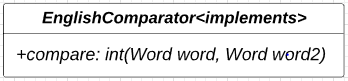
* + getPracticeWords(): Getter method for Practice word.
  + addIntoPracticeWords(): Add a word to the PracticeList.
  + removeFromPracticeWords(): Remove a word from the PracticeList and unfavourite it.
  + savePracticeListToFile(): Save the PracticeList to a JSON File.
  + loadPracticeListFromFile(): Load the PracticeList from a JSON File.
* **Settings -** Where we can set the number of quizzes, change the font size, clear the Practice List, it also allows the user to view help, a description of how to use the application.



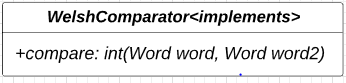
* + getAmountOfWordsInQuiz: Gets how many Words will be used in the Quiz.
  + setAmountOfWordInQuiz: Sets how many words will be used for the Quiz.
  + getLanguageOfQuiz: Get the language preference for the Quiz.
  + setLanguageOfQuiz: Set the Language use for the Quiz.
  + getFontSize: Getter method for the font size.
  + setFontSize: Set the font size for the system to display out of the three options, big, medium and small.
  + saveSettings: Save Settings to JSON File.
  + loadSettings: Load Settings from the JSON File.
* **Word -** Contains all the English, Welsh and Word Type objects. The user can change how the words are sorted using the two comparators later described.



* + Word(): Constructor for the Word class.
  + wordToJSONObject: Takes word object, turns it into a JSON object and stores it in the JSON File.
  + getEnglish: Gets English word.
  + setEnglish: Sets English word.
  + getWelsh: Get Welsh word.
  + setWelsh: Set Welsh word.
  + getWordType: Gets the wordType of word.
  + getWordType: Set the wordType for the word.
  + isFavorited: Gets the Boolean value to see if the word is favourited.
  + setFavorited: Marks the word as favourite or unfavourites its.
  + equals: Compares the English and Welsh parts of both words and returns true if they match, it also removes the «to» in front of the words if they are verbs to make the comparison easier.
* **EnglishComparator<implements> Comparator<Word> -** This defines how the word are compared by their English word. Allows the words to be stored by English.



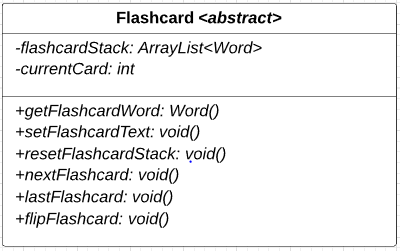
* + compare: Compares if two English words are alphabetically before another or the same.
* **WelshComparator<implements> Comparator<Word> -** This defines how the word are compared by their Welsh word. Allows the words to be stored by Welsh.



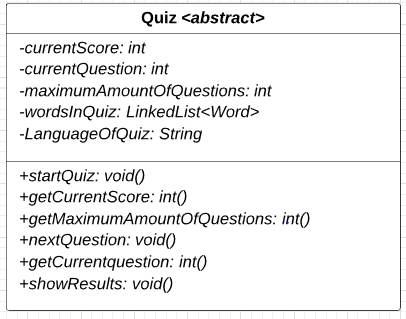
* + compare: Compares if two Welsh words are alphabetically before another or the same.
* **UIHandler -** Handles all the possible java scenes, dealing with the communication between the classes and these. This allows for user interaction, and the sorting of the dictionary and practice list as well as many other important functionalities.



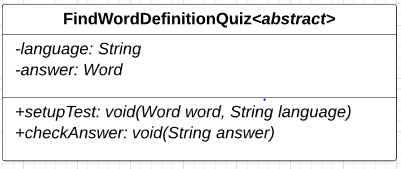
* + setCurrentScene: set the current Scene.
  + getCurrentScene: get the current Scene.
  + loadDictionaryIntoUI: load the Dictionary and display the words into the UI.
  + getTableFromUI: get the Table view from the UI.
  + getUIElement: return a JavaFX Object in the main scene by its FX:id.
  + loadFavoritesIntoUI: load all favourite words into Practice table view.
  + setTableViewProperties: set the Table view properties.
  + initializeUI: initialize the main scene by loading in the home scene.
  + closeMessageBox: close the message Box.
  + createMessageBox: create a Message Box.
  + changeToFlashcardScene: change the Scene to the Flashcard scene.
  + changeFontSizeInUI: change the Font Size for the table view using CSS.
  + getFlashcardInUI: button in the Flashcard scene.
  + changeToSettingsScene: change the scene to the Settings scene.
  + searchButtonClickInUI: search button in UI.
  + clearSearchBar: clear text in the search Bar.
  + openLetterSearchMenu: choose a letter and filter the words in dictionary.
  + sortTableInUI: sort the Table to English or Welsh, and make sure that our button has the corresponding flag graphic set.
  + changeLanguageInUI: change the primary language in UI.
  + showNotification: show a notification.
  + startQuiz: start the Quiz.
  + changeToQuizScene: change scene to the Quiz scene.
  + changeToHomeScene: change scene to the Home Scene
  + changeToRevisionListScene: change scene to the RevisionList scene.
  + changeToRevisionScene: change scene to revision scene.
  + openAddWordToDictionaryUi: add a word user interface.
  + addword: add a new Word to the dictionary.
  + checkWordAddRequirements: check the word being added isn’t invalid.
  + lastFlashcard: go to previous flashcard.
  + nextFlashcard: go to next flashcard.
  + flipFlashcard; flip the flashcard and get the opposite language for the current flashcard.
* **FlashCard <abstract> -** Retrieves the words from the practice list and shows either English or Welsh version of the words one by one, the user views the word and then its translation and can continue to the next word or go back.



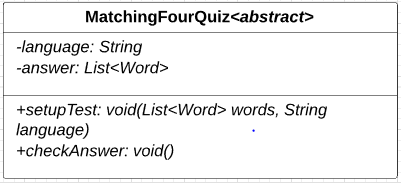
* + getFlashcardWord: get the current word in flashcard.
  + setFlashcardText: update the text on the flashcard to the current word in the flashcard stack.
  + resetFlashcardStack: shuffle flashcard stack once the user has gone through all the words.
  + nextFlashcard: get the next word from the flashcard.
  + lastFlashcard: get the previous word from the flashcard.
  + flipFlashcard: get the translation (other language) from the flashcard.
* **Quiz -** Different types of tests for the user to learn the words from their practice list. Each test takes the words from the practice list and the user must choose the correct answer among a random list of words taken from the dictionary and one of them will be the correct translation. Has the options to continue to the next word or go back.



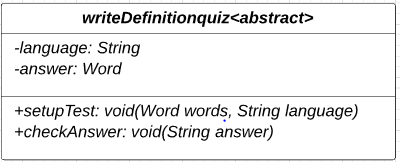
* + startQuiz: Start the Quiz.
  + getCurrentScore: get the current score of the Quiz.
  + getMaximumAmountOfQuestion: get the maximum amount of question for the quiz.
  + nextQuestion: Go to the next question from Quiz.
  + getCurrentQuestion: get the current question from Quiz.
  + showResults: Display the number of question succeed and failed, and the total score of the user.
* **FindWordDefinitionQuiz <abstract> -**Give the word in one language and a list of 6 possible meanings in the other language, so that the user can select the correct translation, feedback is given afterwards.



* + setupTest: create the test to find the word definition from Quiz.
  + checkAnswer: check if the answer correspond to the correct translation of the word.
* **MatchingFourQuiz <abstract> -** Give four words in one language from the practice list and four translations of these words for the user to match, feedback is given afterwards.



* + setupTest: create the test to match 4 words to 4 other corresponding word.
  + checkAnswer: check if the words are matching correctly.
* **WriteDefinitionQuiz <abstract> -** The user is given a word and types in the translation, feedback is given afterwards.



* + setupTest: create test to write the definition of a word in the other language.
  + checkAnswer: check if the answer is the correct translation of the word.

1. DETAILED DESIGN

In this welsh dictionary program, we are going to use algorithm and some data structure to maintain the code.

5.1 Significant algorithms

To give the user a better experience of the quiz, we let the user choose how many words the wish to be tested on in the quiz, they can change this from the settings. However, the user needs to set this before performing the quiz.

When the user starts the quiz, the system will give the user a random set of questions that either ask them match, write or find the definition of a word. The words used will randomly be selected from the practice list and the amount of words selected for the quiz will be added according to the number of tests they want in the quiz. The system will do this by creating a list of words of the amount previously specified, it is first filled up by adding the whole practice list as many times as it will fit into the list. Any leftover space is filled up by selecting random words from the practice list to give an even spread. This list will then be shuffled

User needs to select the submit button for once they have entered their definition quiz and when they are ready to answer the next question. For choosing and finding definition the definition of words, the user needs to choose one of the options to go to the next question. The systems will then check is the question being answered. Once answered there will be a pop-out notification which is in green for answering the correctly or red in colour showing the user that the answer is incorrect and correct answer will be shown for a while. Next question will show immediately after the user has click submit button. These quizzes are produced in a random sequence, the algorithm used to do this will enable the quizzes that are used based on if the number of words being practiced allows the test to be performed (for some there is a minimum ). The program will then produce a random number and a case statement will be used to run one of the tests selected by the random number.

After the user enters the last question, a result window will pop out showing how many words have been answered correctly, how many words that have been answered wrongly, the total amount for the quiz and the average score for the quiz.

5.2 Significant data structure

For data structure, our team has decided to use Linked-List to store and read the words from the dictionary JSON file and practice list JSON file. The reason Linked-List is used because we do not know how much data do, we have from the dictionary JSON file, it does not necessarily need to keep track of the array indexes and it is much easier to work with. We use JASON files to store the practice list and settings as well, so they keep the users previous data, this is because they are easy to read from and provide good formatting.

5 .3 Sequence diagram

A sequence diagram is an interaction diagram that shows the details on how operations are carried out. The purpose of sequence diagram is to model the interaction between objects within a collaboration that realizes an operation.

* Start Up [FR1], Ordering of the list[FR2], Display of words [FR6]

A picture containing drawing

Description automatically generated

* Searching of list [FR3]

A screenshot of a cell phone

Description automatically generated

* Maintaining a practice list [FR4]

Add word to practice list / Mark favourite

A picture containing drawing

Description automatically generated

* Removing word from practice list / Unmark favourite

A picture containing drawing

Description automatically generated

Adding new words to the dictionary [FR5]

A picture containing drawing

Description automatically generated

Reviewing the practice list [FR7]

A screenshot of a cell phone

Description automatically generated

* Flashcards [FR8]

Display flashcard language

A screenshot of a cell phone

Description automatically generated

* Next flashcard word

A screenshot of a cell phone

Description automatically generated

* Previous flashcard word

A screenshot of a cell phone

Description automatically generated

## Flip flashcard get opposite language

A screenshot of a cell phone

Description automatically generated

* Tests on practice words [FR9] and Running tests [FR10]

A picture containing drawing

Description automatically generated

* Remove Word [FR4]

A screenshot of a cell phone

Description automatically generated

**REFERENCES**

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy, B.P. Tiddeman. SE.QA.03. 1.8 Release

[2] Software Engineering Group Projects: General Documentation Standards. C. J. Price. SE.QA.02. 2.3 Release

[3] Software Engineering Group Projects: Design Specification Standards. C. J. Price. SE.QA.05. 2.1 Release

[4] Software Engineering Group Project: Welsh Vocabulary Tutor User Interface Use Case Document. Taavi Karvanen. Mariuz Bialoszewski. SE.QA.UIUC. 1.0

[5] Software Engineering Group Project: Java Coding Standards. C.J.Price. A.McManus. SE.QA.09. 2.0. Release

DOCUMENT HISTORY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| 1.0  \_\_\_\_\_  2.0 | N/A  \_\_\_\_\_  17.0 | 05.03.2020  \_\_\_\_\_\_\_\_\_  20.03.2020 | N/A - original version  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Purpose and Scope changed, missing significant classes added, comments on the methods in interface description added | Kab71  \_\_\_\_\_\_\_\_\_  kmn2 |
| 2.2 | 17.1  18.0  19.0  20.0  22.0  21.0 | 05.05.2020 | Section 1: Purpose and scope changed  Section 2: significant class and requirement table modified  Section 3: UML diagram changed, Component diagram added  Section 4: comment on each class modified and on each method added  Section 5: Significant Algorithms and significant Data structure added, Sequence diagram modified | Kab71  Soj11  Dkc2 |
| 2.3 | 16.0 | 0.6.05.2020 | Config ref updated | Soj11 |