Software Engineering Group Project 20
Test Specification

Author:	Henry Dugmore [hjd3], Kain Bryan-Jones [kab74], Luke Wybar [law39], Marcin Jakob [maj83], Oscar Pocock [osp1], Tom Perry [top1], Waylen Watts [ncw]
Config Ref:	DesignSpecGroup20
Date:	29th March 2020
Version:	1.6
Status:	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to describe the design of the Welsh Vocabulary Tutor program which adheres to the Design Specification Standard Document [2] and General Documentation Standards [1] supplied by the client.

1.2 Scope

This document covers a description of how the Welsh Vocabulary Tutor is designed, including how the software will be broken down. This document should be read by all project members. It is assumed that the reader is already familiar with the Welsh Vocabulary Tutor Requirements Specification [3] and the Design Specification Standards [2].

1.3 Objectives

The objectives of this document are to:

- Identify significant classes.
- Link functional requirements to classes.
- Identify and describe dependencies between modules.
- Determine the public methods of said classes.
- Describe how the classes interact with each other for major operations.
- Identify significant algorithms.
- Identify significant data structures.

2. DECOMPOSITION DESCRIPTION

2.1 Programs in System

Our system is made up of a singular program. This program provides all the functionality required as specified in the Requirements Specification for Welsh Vocabulary Tutor [3] document, including, but not limited to, importing words, adding words manually, practicing words and monitoring achievement through testing the user's word knowledge. The program will also implement all functionality required to pass the functionality tests defined in the Test Specification [4] document.

Our program structure will consist of four key packages including:

- **JSON** – Package that is responsible for handling JSON including the reading/writing of definitions to and from dictionary.json.
- **JavaFX** – Package that contains the JavaFX classes that are all responsible for displaying the UI to the user.
- **SelfAssessment** – Package for holding the classes responsible for generating the user's selfAssessment questions.
- **Test** – Package responsible for holding all the JUnit tests that ensure the program works properly.

2.2 Significant Classes

2.2.1 JSON Package

- **JSONProcessor** – Contains functions responsible for saving and loading to and from the JSON file which will be provided by the user, using the Jackson JSON library.

- **DictionaryEntry** – Class containing all the fields needed for storing dictionary definitions including Welsh and English translations along with its word type and whether it's a practice word or not.

2.2.2 JavaFX Package

- **Application** – Programs main class that contains a list of the loaded dictionary definitions and is responsible for running the application.
- **SharedCodeController** – Abstract class that contains all the shared FXML elements between the different controller classes including the sliding menu and the test score counter, to reduce code duplication. This will be extended by all the controller classes.
- **ScreenSwitch** – Class that contains all the scenes for the JavaFX user interface and will be responsible for initiating the transition to new ones.

2.2.3 SelfAssessment Package

- **Question** – Abstract class contains the basic information that all the shared information between the types of test questions including the questions' correct answers and possible answers. All question classes will extend this class.
- **WordMatchQuestion –** Class that contains all the details needed for the 'Match the Meanings' question type, including the 4 different practice definitions. This class will be used by the AssessmentGenerator and extends the Question class.
- **WordEnterQuestion -** Class that contains all the details needed for the 'Translation' question type, including the practice definition that is being tested. This class will be used by the AssessmentGenerator and extends the Question class.
- **SixMeaningsQuestion –** Class that contains all the details needed for the 'Six Meanings' question type, including the correct answer along with the five other possible answers. This class will be used by the AssessmentGenerator and extends the Question class.
- **AssessmentGenerator** – Class that contains methods to create a randomised list of questions that will contain a random distribution of question types.

2.2.4 Test Package

- **JSONTest** – Class that contains methods which will be used to test that the JSON package classes are correctly loading and saving to and from the JSON file.
- **JavaFXTest** – Class that contains methods to test that the application class is correctly storing the full list of dictionary definitions.
  Furthermore, this class will also test that the elements such as the sliding menu and score counter are working as intended, along with testing that scenes are ending and transitioning correctly when applicable.
- **SelfAssesmentTest** – This class will test that the lists pulled in the selfAssessment package are indeed random, while also pulling the matching data from the dictionary.

2.3 Table Mapping Requirements to Classes

Functional Requirement	Classes implementing
FR1 Startup	Application, LaunchScreenController, SharedCodeController, JSONProcesser, DictionaryEntry
FR2 Ordering of the list	LaunchScreenController, DictionaryEntry
FR3 Searching of list	LaunchScreenController, DictionaryEntry
FR4 Maintaining a practice list	Application, LaunchScreenController,

	DictionaryEntry
FR5 Adding new words to the dictionary	Application, AddWordScreenController
FR6 Display of words	Application, LaunchScreenController
FR7 Reviewing the practice list	Application, PracticeListScreenController
FR8 Flashcards	Application, FlashcardScreenController
FR9 Tests on practice words	Application, SixMeaningsQuestionScreenController, WordEnterQuestionScreenController, WordMatchQuestionScreenControler
FR10 Running tests	Application, TestSelectionScreenController, SixMeaningsQuestionScreenController, WordEnterQuestionScreenController, WordMatchQuestionScreenControler

3. DEPENDENCY DESCRIPTION

3.1 Component Diagram

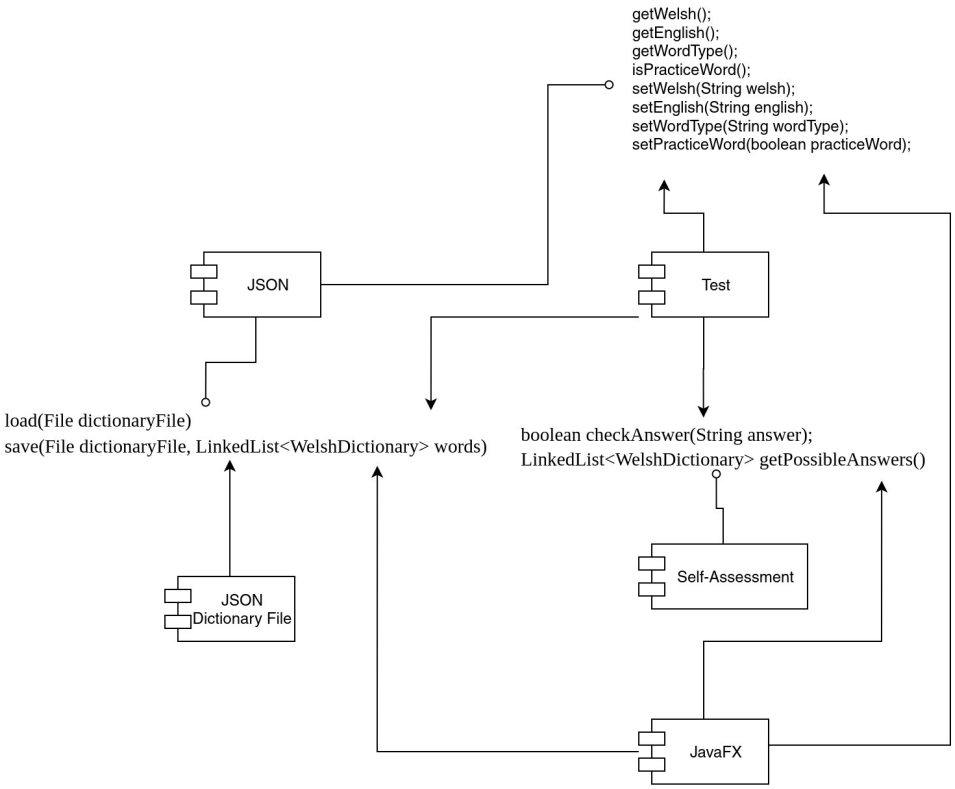


Figure 1: Component diagram of the Welsh Vocabulary App

4. INTERFACE DESCRIPTION

4.1 JSON Package

4.1.1 JSONProcessor

Class that will handle the program's JSON related work, including the loading/saving of the files.

- **public LinkedList<** DictionaryEntry **> load(File dictionaryFile)** - Method for loading the list of dictionary definitions from a JSON file.
- **public void save(File dictionaryFile, LinkedList<** DictionaryEntry **> words)** - Method for saving the list of dictionary definitions to a JSON file.

4.1.2 DictionaryEntry implements Comparable< DictionaryEntry >

Class that will hold each word's definition with all the necessary fields.

- **public DictionaryEntry()** - Default constructor for DictionaryEntry.
- **public DictionaryEntry(String english, String welsh, WordType wordType, Boolean practiceWord)** - Constructor for DictionaryEntry that includes a full list of parameters.
- **public String getWelsh()** - Getter method for the dictionary objects welsh variable.
- **public String getEnglish()** - Getter method for the dictionary objects english variable.
- **public String getWordType()** - Getter method for the dictionary objects word type variable.
- **public boolean isPracticeWord()** - Getter method for the dictionary objects practiceWord variable.
- **public void setWelsh(String welsh)** - Setter method for the dictionary objects welsh variable.
- **public void setEnglish(String english)** - Setter method for the dictionary objects english variable.
- **public void setWordType(String wordType)** - Setter method for the dictionary objects word type variable.
- **public void setPracticeWord(boolean practiceWord)** - Setter method for the dictionary objects practiceWord variable.
- **@Override public boolean equals(Object obj)** - Equals method for checking if two dictionary objects are equal.
- **@Override public int compareTo(Object obj)** - Method for comparing two DictionaryEntry objects, used for sorting the list of definitions alphabetically.

4.2 SelfAssessment Package

4.2.1 Question

Abstract class that holds general information such as each questions possible answers and also the correct answer.

- **public boolean checkAnswer(String answer)** - Method to check whether a given answer matches the question's correct answer.
- **public LinkedList<DictionaryEntry> getPossibleAnswers()** - Getter method for the question objects possible answers.

4.2.2 AssessmentGenerator extends Question

Class that contains methods to create a randomised list of Assessment that will contain a random distribution of question types.

- **public LinkedList<Question> generateAssessment(LinkedList<DictionaryEntry> words)** - Method that will generate a randomized list of questions consisting of random distribution of questions types, using the dictionary's practice words as the parameter.
- **public LinkedList<Question> generateWordMatch(LinkedList<DictionaryEntry>) -** Method that will generate a list of questions that are the type 'Match The Meanings', using the dictionary's practice words as the parameter.
- **public LinkedList<Question> generateSixMeanings(LinkedList<DictionaryEntry>) -** Method that will generate a list of questions that are the type '6 Meanings', using the dictionary's practice words as the parameter.
- **public LinkedList<Question> generateWordEnter(LinkedList<DictionaryEntry>) -** Method that will generate a list of questions that are the type 'Translation', using the dictionary's practice words as the parameter.

4.2.3 WordEnterQuestion extends Question

- **public WordEnterQuestion (DictionaryEntry correctAnswer)** - Constructor for WordEnterQuestion that takes a DictionaryEntry object that is being tested on as the parameter.

4.2.4 WordMatchQuestion extends Question

- **public WordMatchQuestion (DictionaryEntry[] correctAnswers)** - Constructor for WordMatchQuestion that takes four DictionaryEntry objects that are being tested on as the parameters.

4.2.5 SixMeaningQuestion extends Question

- **public SixMeaningQuestion (DictionaryEntry correctAnswer, LinkedList<DictionaryEntry> dictionary)** - Constructor for SixMeaningQuestion that takes one DictionaryEntry object that is being tested along with the full list of words which will be used to generate randomized possible answers as the parameters.

4.3 JavaFX package

4.3.1 Application

Programs main class where the program will start from. This class will also hold the programs dictionary definitions.

- **Main()** – runs app.

4.3.2 SharedCodeController

Abstract class that will hold all of the repeated information between controllers including common FXML elements that will be derived by the controllers. This could include the sliding menu options and user test scores.

4.3.3 ScreenSwitch extends SharedCodeController

- **SceneEnum**

  SceneEnum is a enumeration type for storing the different types of scenes. The different possible values are 'addWordScene', 'dictionaryScene', 'flashcardScene', 'praciceListScene', 'matchMeaningScene', 'sixMeaningsScene', 'translationScene'.

- **public void swap(SceneEnum newScene)** - Method that is responsible for the switching between JavaFX, with it taking the new scene's name as a parameter.

4.4 Test Package

4.4.1 JSONTest

Class that contains methods which will be used to test that the JSON package classes are correctly loading and saving to and from the JSON file.

- **@test public void testLoad()** - JUnit test to check that the JSON file has been correctly loaded.
- **@test public void testSave()** - JUnit test to check that any changes to the list of definitions are updated and saved to the JSON file accordingly.

4.4.2 JavaFXTest

Class that contains methods to test that the application class is correctly storing the full list of dictionary definitions. This class will also test that the elements such as the sliding menu and score counter are working as intended, along with testing that scenes are ending and transitioning correctly when applicable.

- **@test public void testDefinition()** - Tests to confirm that the dictionary definitions loaded match to an identical base set.
- **@test public void testScoreCounter()** - Test to confirm that the user score counter correctly increases by increments on one.
- **@test public void testFindWord() –** A preset search test to confirm that words are being searched for correctly.
- **@test public void testAddWord() -** A test to check that a new word is correctly added and saved to the JSON file.
- **@test public void testRemoveWord() -** A test to check that the JSON file is correctly updated when a word is removed.

4.4.3 SelfAssessmentTest

This class will test that the lists pulled in the selfAssessment package are indeed random, while also pulling the matching data from the dictionary.

- **@test public void testRandomReturn()** - Test to confirm that the random number return is working correctly.
- **@test public void testAvailableSelfAssessment() -** Test to check and confirm that the game types are either made available or are locked off depending on the number of practice list questions.
- **@test public void testUserAnswer() –** Test that will check that an input by a user is correctly checked to the correct answer.

5. DETAILED DESIGN

5.1 Sequence Diagrams

5.1.1 Use Case 1 View dictionary

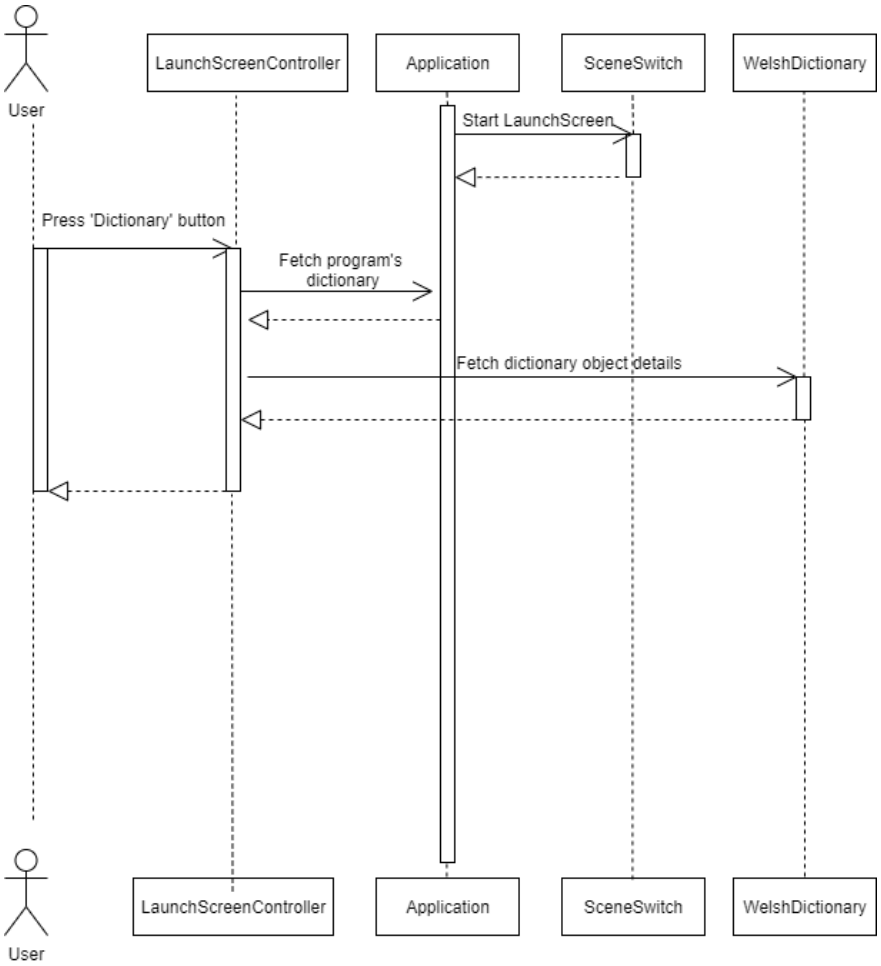


Figure 2: Sequence diagram for displaying the Dictionary

5.1.2 Use Case 2 Search for a word

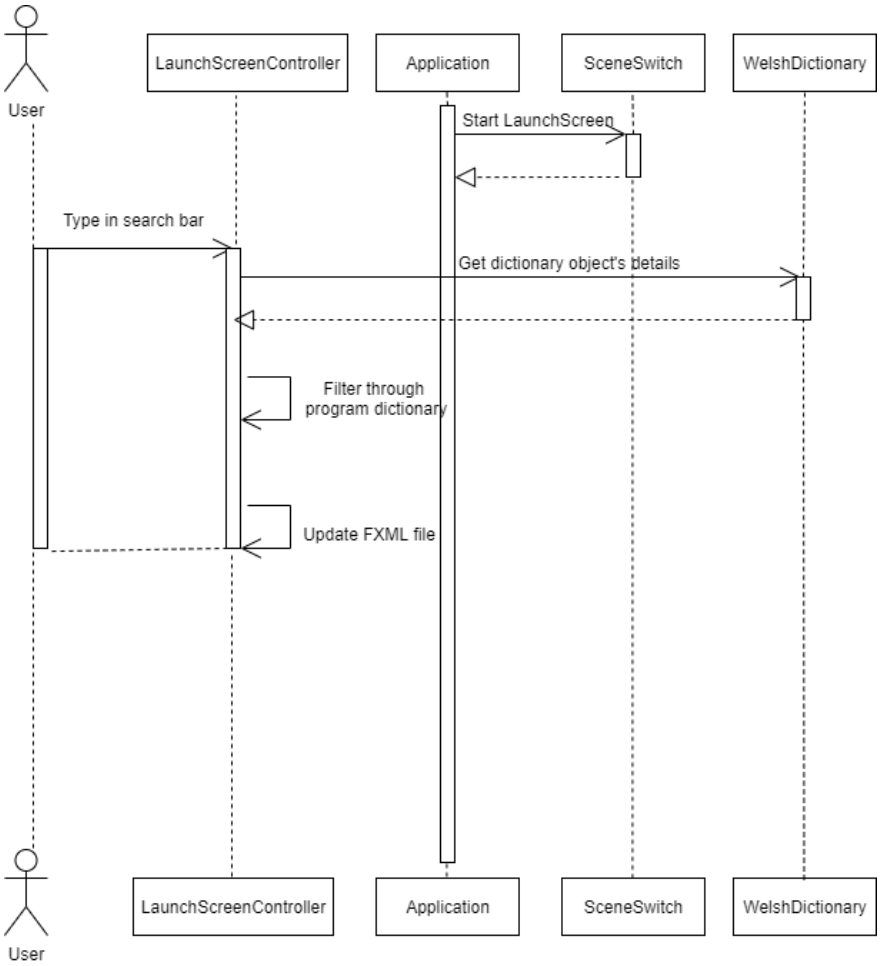


Figure 3: Sequence diagram for performing word search

5.1.3 Use Case 3 View practice list

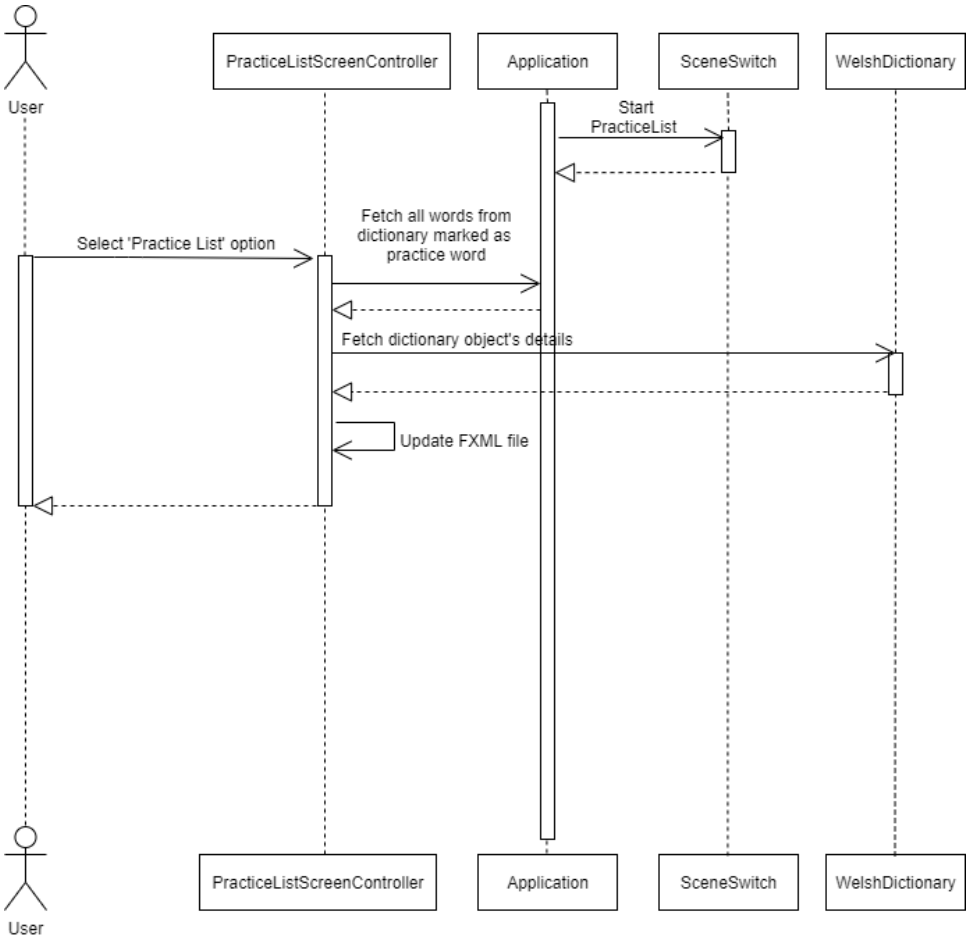


Figure 4: Sequence diagram for displaying the practice list

5.1.4 Use Case 4 Modify the practice list

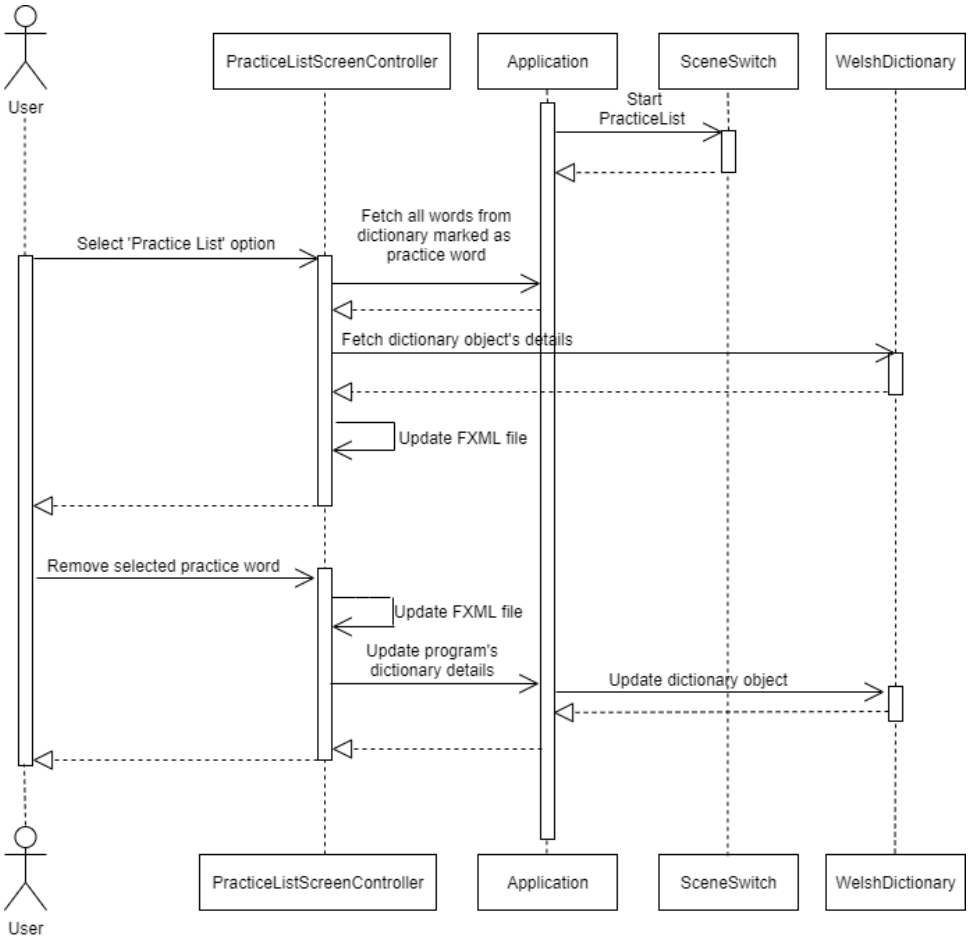


Figure 5: Sequence diagram for removing words from the practice list

5.1.5 Use Case 5.1 Start 'Match The Meaning' test

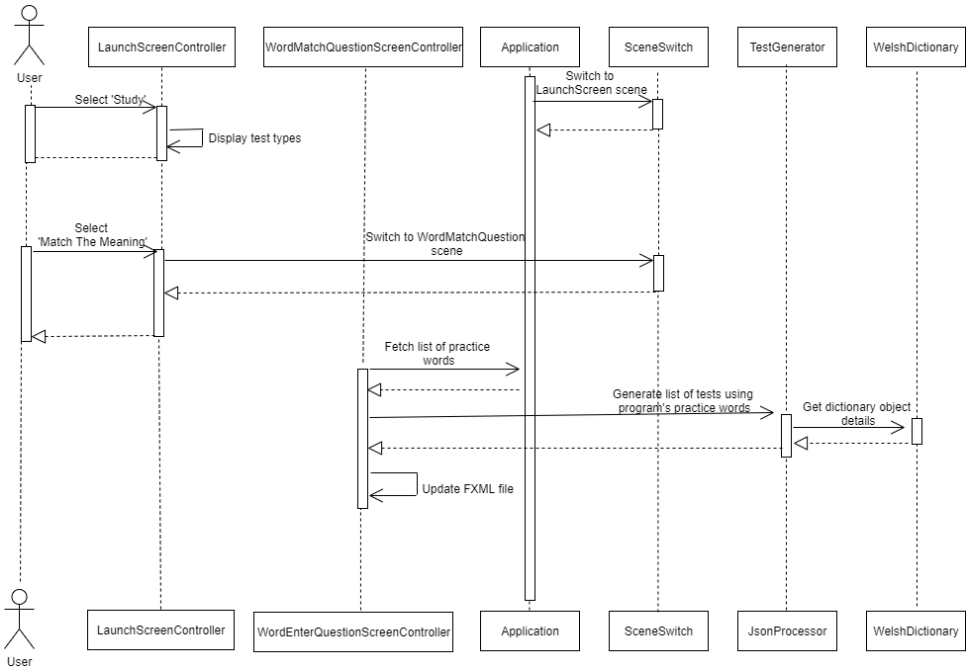
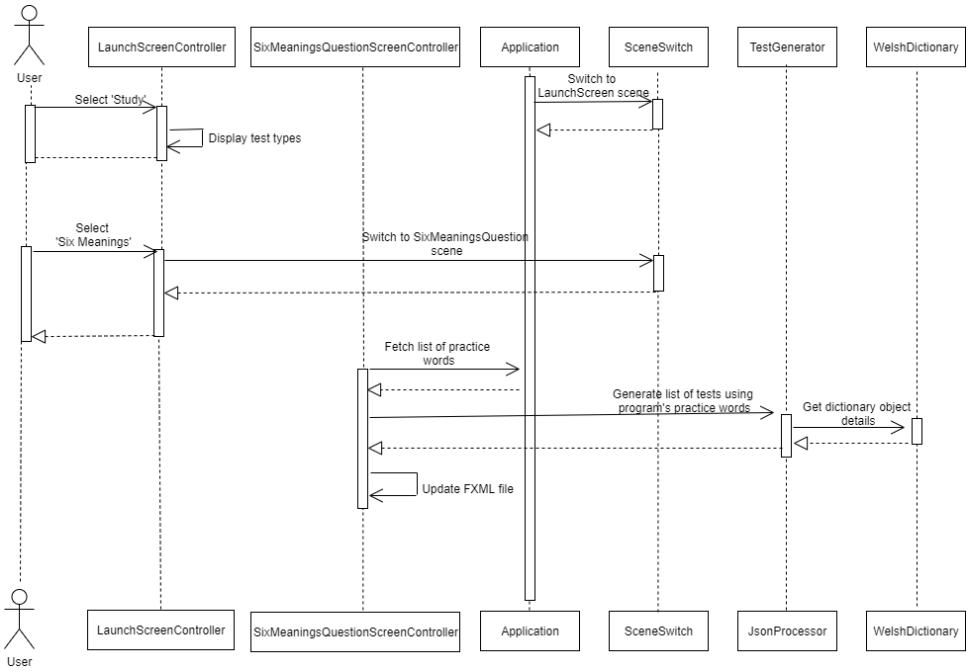


Figure 6: Sequence diagram for the 'Match The Meaning' test

5.1.6 Use Case 5.2 Start ‘6 Meanings’ test



5.1.7 Use Case 5.3 Start ‘Translation’ test

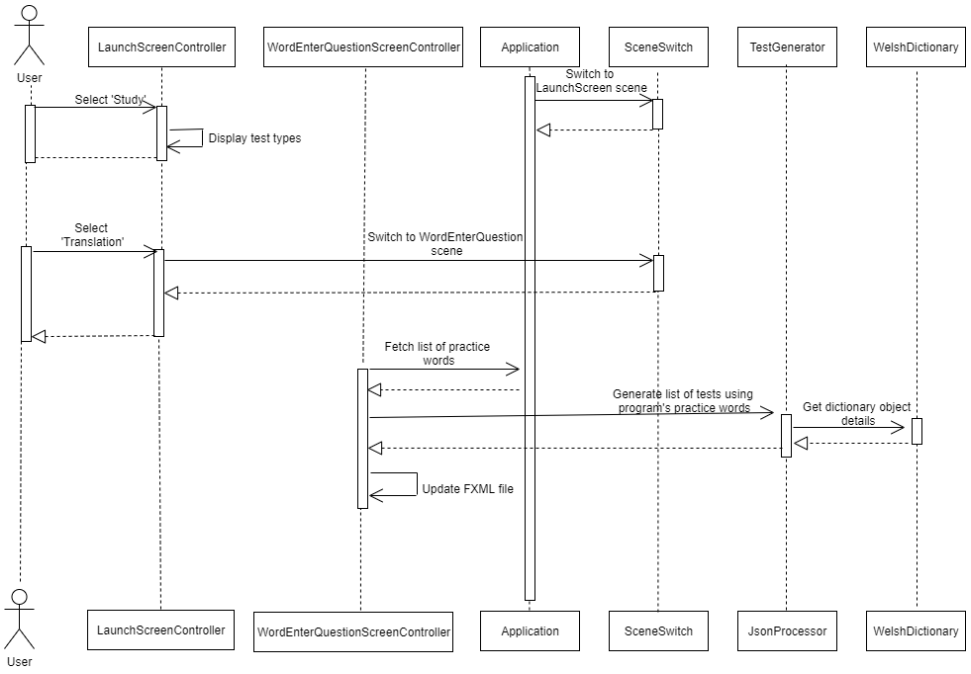


Figure 8: Sequence diagram for the ’Translation’ test

5.1.8 Use Case 6 View flashcards

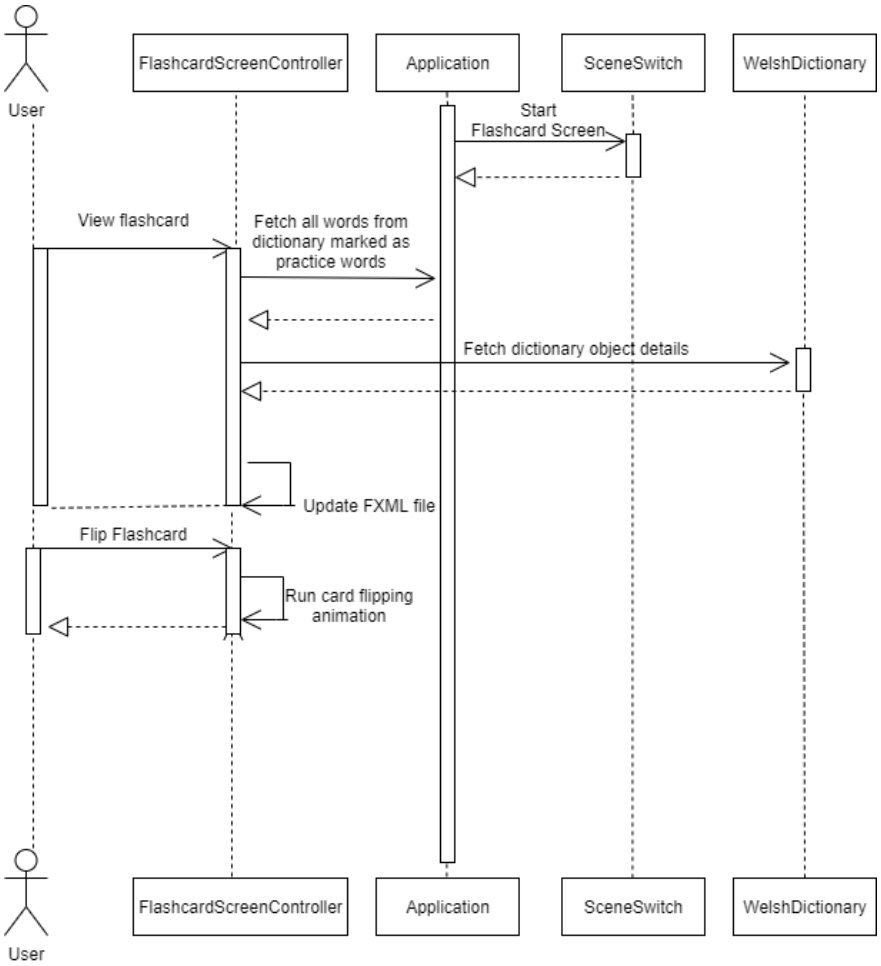


Figure 9: Sequence diagram for loading the dictionary list

5.1.9 Use Case 7 Add a new word

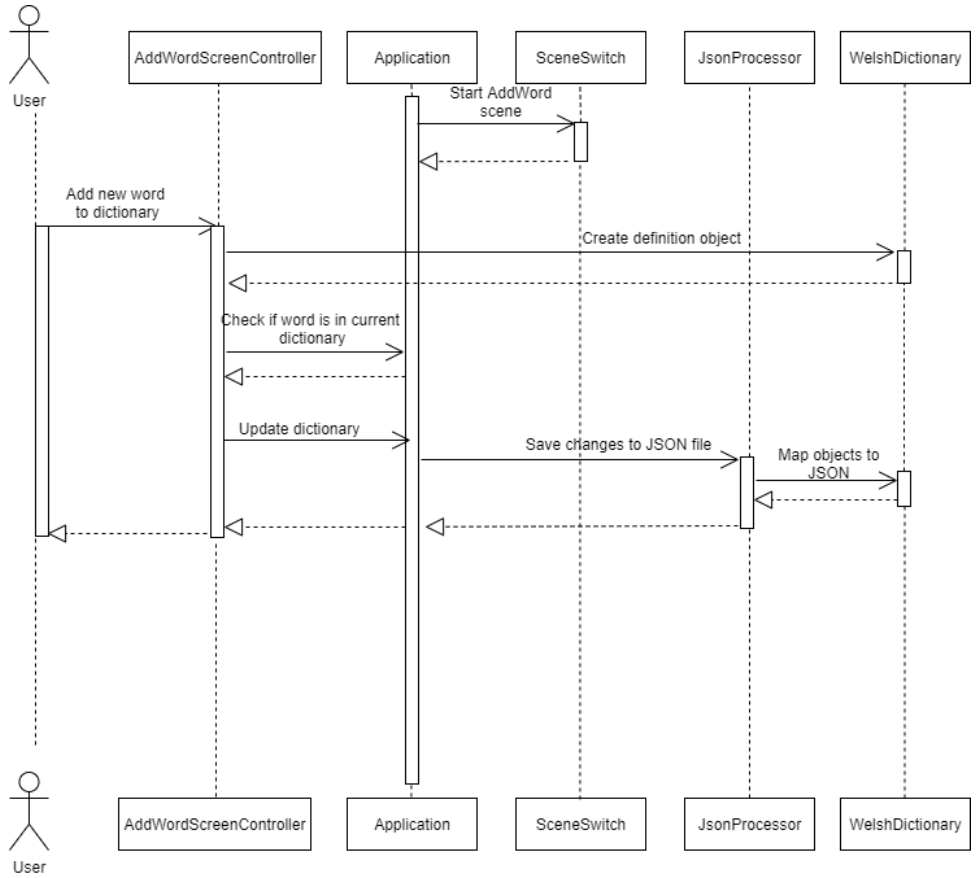


Figure 10: Sequence diagram for adding new words

5.1.10 Use Case 8 Change word ordering

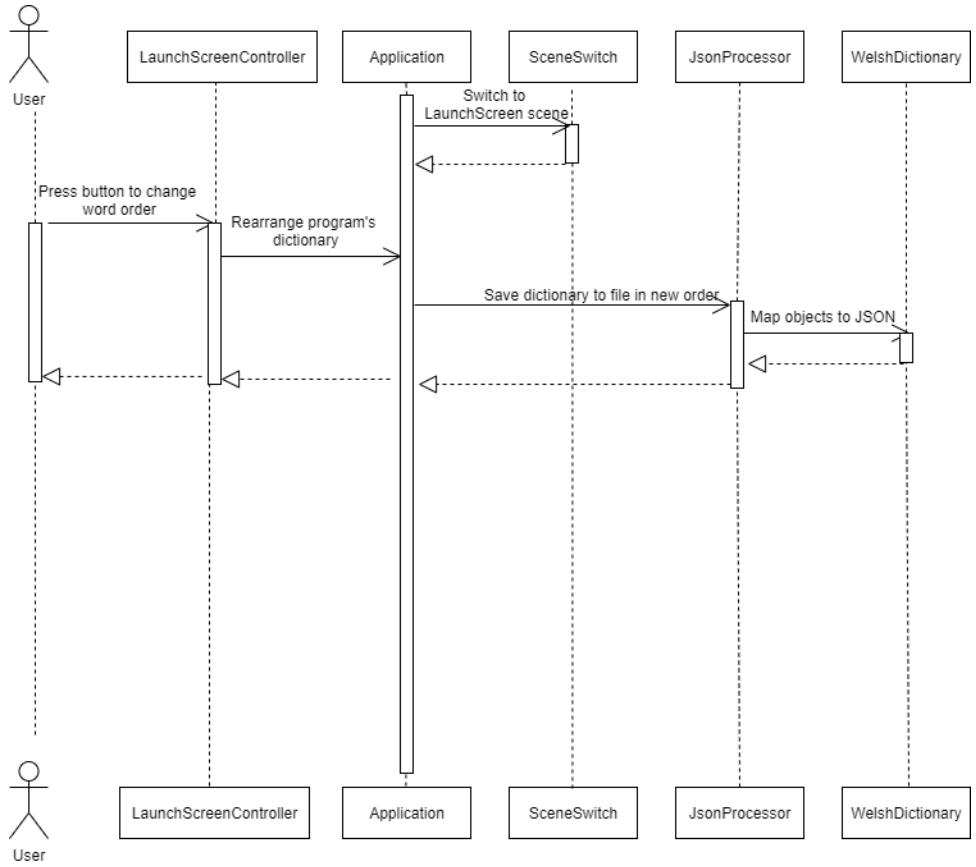


Figure 11: Sequence diagram for changing the ordering of words

5.2 Significant Algorithms

5.2.1 JavaFX screen switching algorithm

All JavaFX screens will be loaded in at runtime and switching will be achieved by calling a method in the JavaFX control class which takes the name of the requested screen as an enumeration and handles preparing the screen and finally puts it on the stage.

5.2.2 Live-searching algorithm

The live-search shall be achieved by modifying the equals method to consider the length of input strings, therefore allowing search to match strings which haven't been fully entered yet.

5.2.3 Adding words algorithm

Each new word added will create a new DictionaryEntry object, constructed using the Welsh, English and word type to populate the instance variables. This is then added to a list data structure, which is then used by other modules of the program for displaying, practicing and testing words with the user.

5.2.4 Saving algorithm

The saving shall be performed at the closure of the program, this will be completed through the use of the Jackson library, this provides a simple way of encoding the data in the program into JSON. This is then written out to a flat file.

5.2.5 Loading algorithm

The loading algorithm will run on request of the user through a button press. It will use JavaFX to open a filePicker, and when the user picks a file with valid JSON, this will be loaded in and mapped to DictionaryEntry objects by the Jackson library, these objects are then added to the list data structure.

5.3 Significant Data Structures

5.3.1 Linked Lists:

Currently the program works with DictionaryEntry objects, which store the Welsh translation of the word, its English translation and the word type (verb, masculine noun, etc). These objects are linked lists which would point to the next object in the dictionary, i.e. it would have the next word down adjacent to the object.

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: Design Specification Standards. C. J. Price, SE.QA.05. 2.1 Release

[3]Software Engineering Group Projects: Welsh Vocabulary Tutor Requirement Specifications. C. J. Price, SE.QA.CSRS. 1.1 Release

[4]Software Engineering Group Project 20: Test Specification. N. C. Watts, H. J. Dugmore, TestSpecGroup20. 1.0 Release

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	27/03/2020	Created document based on CP's template.	OP
0.2	N/A	30/03/2020	Corrected spelling mistakes and formatting.	BC, KB, LW, OP, TP
1.0	N/A	31/03/2020	Corrected grammatical issues, and font sizes.	OP
1.5	12	29/04/2020	Refactor of WelshDictionary -> DictionaryEntry.	NCW