Software Engineering Group Project 20
Design Specification

Author:	Henry Dugmore [hjd3], Kain Bryan-Jones [kab74], Luke Wybar [law39], Marcin Jakob [maj83], Oscar Pocock [osp1], Tom Perry [top19], Waylen Watts [ncw]

Config Ref:	DesignSpecGroup20
Date:	5th May 2020
Version:	1.7
Status:	Review

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 self assessment 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. It also contains a list of practice words.
- **SharedCodeController** – Abstract class that contains all the shared FXML elements between the different controller classes including the sliding menu, 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 which contains basic information shared between all the types of test questions including the questions' check for correct answers method and a local count for the number of correct and incorrect answers for a single question. All question classes will extend this class.
- **MatchTheMeaningQuestion –** 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.
- **TranslationQuestion -** 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. It will also track the users correct and total answers attempted for the whole assessment of multiple questions.

2.2.4 test Package

- **AssessmentGeneratorTest** – This class will test that the lists pulled in the selfAssessment package are indeed random, while also pulling the matching data from the dictionary.
- **DictionaryEntryTest** - This class will perform a number of tests to ensure the DictionaryEntry datatype can reliably and accurately represent dictionary entries.

- **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.
- **QuestionTest** - Class that tests the different types of questions available to ensure the user is accurately marked for their work.

2.3 Table Mapping Requirements to Classes

Functional Requirement	Classes implementing
FR1 Startup	Application, ScreenSwitch, SharedCodeController, JSONProcessor, DictionaryEntry, DictionaryController
FR2 Ordering of the list	DictionaryController, DictionaryEntry, PracticeListController, SharedCodeController, Application
FR3 Searching of list	DictionaryController, DictionaryEntry, PracticeListController, Application, SharedCodeController
FR4 Maintaining a practice list	DictionaryController, PracticeListController, SharedCodeController, DictionaryEntry, Application, AddWordController
FR5 Adding new words to the dictionary	Application, AddWordController, DictionaryEntry, SharedCodeController
FR6 Display of words	DictionaryController, PracticeListController, SharedCodeController, DictionaryEntry, Application
FR7 Reviewing the practice list	PracticeListController, SharedCodeController, DictionaryEntry, Application
FR8 Flashcards	Application, FlashcardController, SharedCodeController, DictionaryEntry
FR9 Tests on practice words	AssessmentGenerator, Question, TranslationController, SixMeaningsController, MatchTheMeaningController, SharedCodeController, TranslationQuestion, SixMeaningsQuestion, MatchTheMeaningQuestion
FR10 Running tests	Application,

	SixMeaningsController, TranslationController, MatchTheMeaningController AssessmentGenerator

3. DEPENDENCY DESCRIPTION

3.1 Component Diagram

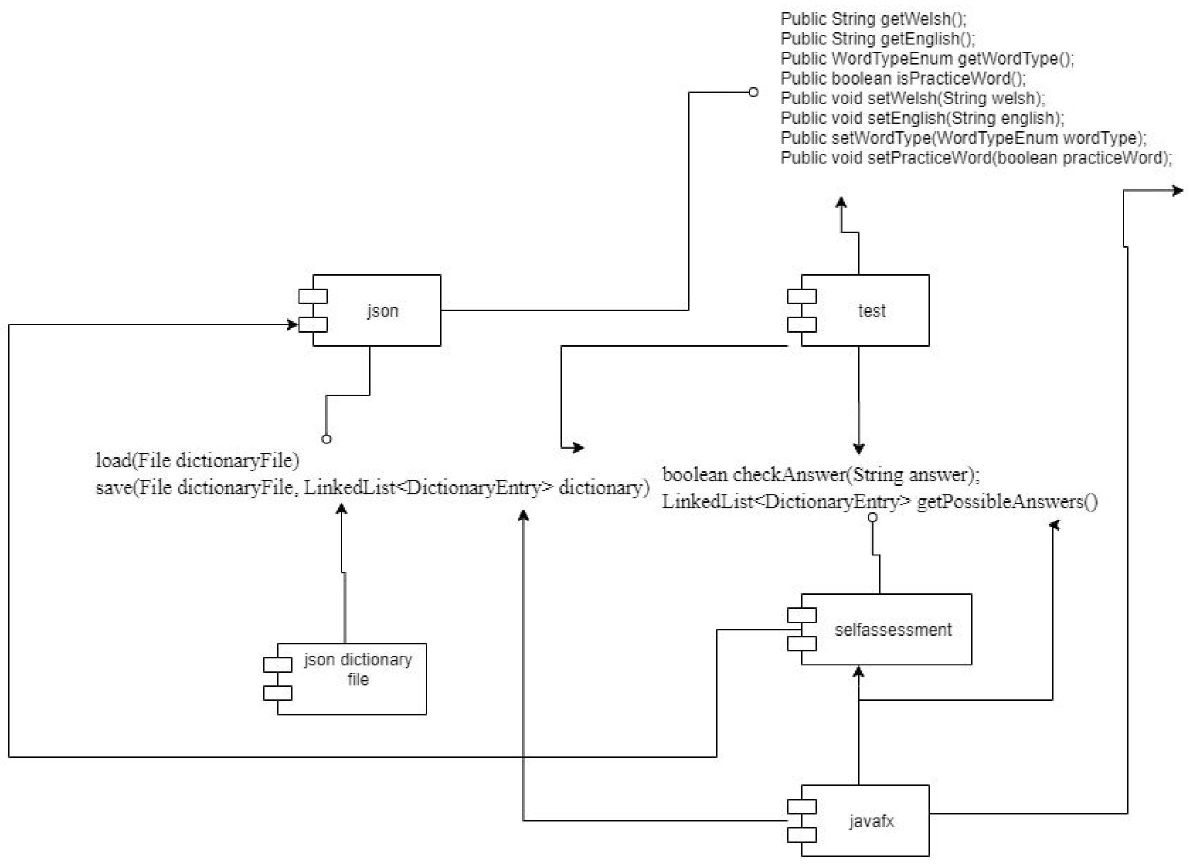
Public String getWelsh();
Public String getEnglish();
Public WordTypeEnum getWordType();
Public boolean isPracticeWord();
Public void setWelsh(String welsh);
Public void setEnglish(String english);
Public setWordType(WordTypeEnum wordType);
Public void setPracticeWord(boolean practiceWord);
json
test
load(File dictionaryFile)
save(File dictionaryFile, LinkedList<DictionaryEntry> dictionary)
boolean checkAnswer(String answer);
LinkedList<DictionaryEntry> getPossibleAnswers()
json dictionary file
selfassessment
javafx

Figure 1: Component diagram of the Welsh Vocabulary App

4. INTERFACE DESCRIPTION

4.1 json Package

4.1.1 JSONProcessing

A class that handles the import and export of Json-formatted files, following the schema set out in SE.QA.CSRS DC3

- **public LinkedList< DictionaryEntry > readInJason(File file) –** Method to read in a Json file formatted in the schema set out in SE.QA.CSRS DC3
- **public void writeOutJSON(File dictionaryFile, LinkedList<** DictionaryEntry **> words)** - Method to write out a Json file formatted in the schema set out in SE.QA.CSRS DC3

4.1.2 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.

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.
- **public static void CheckAnswer(ArrayList<DictionaryEntry> listOfCorrectQuestions, ArrayList<String> listOfAnswers, boolean isEnglish) -** Function that checks the answers of questions. Checks whether they're right and uses an object instance of StringBuilder to build an appropriate sentence to present to the user to give them their feedback. E.g. "Apple is the English for Afal is correct"
- **public void showFeedback() -** Function for giving users positive or negative feedback for when they answer a question during an assessment.
- **public static void resetScore()** - Resets the score to 0 for the next test.

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 static LinkedList<Question> generateAssessment(LinkedList<DictionaryEntry> dictionary)** - 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 Question generateSixMeanings(LinkedList<DictionaryEntry> dictionary) -** Method that will generate a list of questions that are the type '6 Meanings', using the dictionary's practice words as the parameter.
- **public static Question generateMatchMeaning(LinkedList<DictionaryEntry> practiceList) –** Method that will generate a list of questions that are of the type 'Match The Meanings' using the dictionary's practice words as the parameter.
- **public static Question generateTranslationTest(LinkedList<DictionaryEntry> practiceList) –** Method that will generate a list of questions that are the type 'Translation', using the dictionary's practice words as the parameter.
- **public static void gotToNextQuestion() -** Method usescurrentAssessment as pointer to go to next question in assessment list.
- **public static void reset() -** Method for resetting assessment to default stage.

4.2.3 TranslateQuestion extends Question

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

4.2.4 MatchTheMeaningQuestion 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.
- **public ArrayList<DictionaryEntry> getCorrectAnswer() -** Function to retrieve the correct answer to a SixMeaningsQuestion.

4.3 javafx package

4.3.1 AddWordController extends SharedCodeController

A class that handles the keyboard and mouse input and interaction for the 'Add Word Page' which is defined by 'addword.fxml'.

- **public void addCharXX(ActionEvent actionEvent)** - Method that adds whatever value XX will be(e.g. "ch", "th", "ph") to the welsh text field and runs when the user clicks the XX button on the add word screen.
- **public void specialCharX(ActionEvent actionEvent)** – There are four different 'specialChar' methods which append the values of specialChar '1-4' to the welsh word.

4.3.2 DictionaryController extends SharedCodeController

A class that handles the keyboard and mouse input and interaction for the 'Dictionary Page' which is defined by 'dictionary.fxml'.

- **public void initialize()** - Initializes the table of dictionary entries. An observable list of DictionaryEntries is loaded from the Application class into a local instance of ObservableList. It also sets up Lambda expressions related to live searching functionality and the display of DictionaryEntries.

- **private void switchLangSort() -** Method to switch the language used to sort the dictionary list. If currently sorted by English, this will change the sort to by Welsh. If currently sorted by Welsh, this will change the sort to by English.
- **private void switchAlphaSort() -** Method to switch the alphabetical order used to sort the dictionary list.

4.3.3 FlashcardController extends SharedCodeController

A class that serves as the controller for the programs Flashcard JavaFX scene, handling all of its events and attributes. This scene is defined as "flashcard.fxml".

- **private void initialize() -** Method that initializes ‘flashcard.fxml’ by setting up the icons and text. This method is called automatically whenever the flashcard scene starts.
- **private void handleFlashcardClick() -** Event that rotates the scenes flashcard using RotateTransition whenever the user clicks the flashcard.
- **private void handlePreviousCard() -** Event that switches to the previous flashcard whenever the user clicks the ‘leftArrow’ icon.
- **private void handleNextCard() -** Event that switches to the next flashcard whenever the user clicks the ‘right-arrow’ icon.
- **private void updateCounter() -** Method that updates the onscreen counter of the current flashcard.
- **private RotateTransition RotateCard(Node card) -** Method that creates a RotateTransition for flipping the card 180 degrees.

4.3.4 MatchTheMeaningController extends SharedCodeController

A class that generates questions and checks for answers to match the meaning test.

- **public void setWords(ArrayList<DictionaryEntry> questions, ArrayList<Integer> orderList)** – Sets chosen words from the dictionary on the scene.
- **public void checkAnswers()** - Checks if answers from users are correct.
- **private void initialize() -** see 4.3.2

4.3.5 PracticeListController extends SharedCodeController

A class that handles the keyboard and mouse input and interaction for the 'Dictionary Page' which is defined by 'dictionary.fxml'.

- **public void initialize()** - see 4.3..2.
- **private void switchLangSort()** - Method to switch the language used to sort the dictionary list.
- **private void switchAlphaSort()** - Method to switch the alphabetical order used to sort the dictionary list.

4.3.6 SixMeaningsController

- **public void setWords(ArrayList<DictionaryEntry> questions, ArrayList<Integer> orderList)** - Method that sets up the SIxMeanings question onto the screen. It firstly starts by checking the type of question and displaying the possible answer based off of this.
- **public void checkAnswers()** - Method checks the answer the user has submitted against the questions correct answer. This works by passing in the users ‘WordCounterPart’ answer with the correct answer into the Question class which does the checking before moving onto the next question.
- **private void initialize() -** see 4.3.2
- **public void answerX() -** Event that runs when the user clicks the Xth answer from the six options. This sets the ‘wordCounterPart’ to the value in ‘possibleAnswerX’before checking the answer.

4.3.7 TranslationController

- **public void specialCharX(ActionEvent actionEvent)** – see 4.3.1.
- **private void initialize() -** see 4.3.2

- **void translateWord() -** Takes the word the user inputs and compares it to the correct answer using the checkAnswer function in the QuestionClass.

4.3.8 Application

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

- **Main()** – runs app.

4.3.9 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.

- **public void setup() -** Method that sets up the program's menu in each of the controllers initialising the icons and text.
- **private void initializeIcons() -** Method that sets up all of the menu icons by setting them to the images stored within the resources file.
- **private void initializeMenuText() -** Method that sets up all of the menu's text by setting them to their desired text when the menu is expanded.
- **private void disableMenuText() -** Method that disables the menu's text when the menu is collapsed by setting their text to nothing.
- **private void expandMenuClick() -** Event that collapses or expands or expands the menu whenever the 'expandMenuicon' is clicked by the user. The method determines the menu current state by looking at the value of 'sideBarWidth' and uses that to decide whether the menu needs to expand to 230 and initialise the menu text or collapse to 50, disabling menu text.
- **private void dictionaryIconClick() -** Event to switch scenes to 'dictionary.fxml' when the menu's 'dictionaryIcon' icon is clicked.
- **private void practiceListIconClick() -** Event to switch scenes to 'practicelist.fxml' when the menu's 'practiceListIcon' is clicked.
- **private void flashcardIconClick() -** Event to switch scenes to 'flashcard.fxml' when the menu's 'practiceListIcon' icon is clicked. This method checks to see if practiceList is empty before switching in order to avoid a NullPointerException in the flashcard scene.
- **private void studyIconClick() -** Event to generate an assessment using AssessmentGenerator when the menu's 'studyIcon' icon is clicked.
- **private void addWordIconClick() -** Event to switch scenes to 'addword.fxml' when the menu's 'addWordIcon' icon is clicked.

4.3.10 ScreenSwitch extends SharedCodeController

- **SceneType** - SceneType is an enumeration type for storing the different types of scenes. The different possible values are 'addWordScene', 'dictionaryScene', 'flashcardScene', 'praciceListScene', 'matchMeaningScene', 'sixMeaningsScene', 'translationScene'.
- **public ScreenSwitch(Stage stage)** - This constructor is used by Application to pass control to the stage. It will also display the launch scene on the stage to the user. Change the Scene loaded here to change the launch screen.
- **public static void swap(SceneType newScene)** – Method that is responsible for switching between JavaFX, with it taking the new scene's name as an enum as a parameter.

4.4 test Package

4.4.1 DictionaryEntryTest

A test class that contains methods testing that DictionaryEntry works as intended.

- **void testPracticeWordFalse()** - Tests whether the default constructor sets isPracticeWord to false upon declaration of a new DictionaryEntry.
- **void testAllSettersAndGetters()** - Tests whether the setters and getters of the DictionaryEntry class work as intended.
- **void testEqualsTruePossitive()** - A true-positive test for the equals method in DictionaryEntry.
- **void testEqualsTrueNegative()** - A true-negative test for the equals method in DictionaryEntry.

4.4.2 AssessmentGeneratorTest

A test class for various tests regarding AssessmentGenerator.

- **public void testNumOfAssessment()** - JUnit test to see if the right amount of tests is generated.

4.4.3 QuestionTest

- **void testCheckRightAnswerTranslationOrSixMeanings()** - JUnit tests that the correctAnswers variable increments when a user gets a right answer when doing either a Translation or SixMeanings test.
- **void testCheckWrongAnswerTranslationOrSixMeanings()** - JUnit Tests that the wrongAnswers variable increments when a user gets a wrong answer when doing either a Translation or SixMeanings test.
- **void testCheckRightAnswerMatchMeaning()** - Tests that the correctAnswers variable increments when a user gets a right answer when doing either a MatchTheMeaning test.
- **void testCheckWrongAnswerMatchMeaning()** - Tests that the wrongAnswers variable increments when a user gets a wrong answer when doing either a MatchTheMeaning test.
- **void resetScore()** – JUnit test to check whether the Question class method 'resetScore' works as intended.

4.4.4 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.

- **@BeforeAll public static void setupTest() -** Setup method that is run before all of the tests, setting up a test list of DictionaryEntry that is saved to a JSON test file.
- **static void deleteFile()** -Method that is run after the JUnit tests have finished to remove the JSON test file from the program.
- **void testLoad()** - JUnit test to check that the JSON file has been correctly loaded. This works by loading the test file and check
- **public void testSave() -** JUnit test to check that changes to the list of definitions are updated and saved to the JSON file accordingly. This is done by adding a new item to the JSON test list and saving it to the file before reloading it to check if the loaded list matches the updated test list.

5. DETAILED DESIGN

5.1 Sequence Diagrams

5.1.1 Use Case 1 View dictionary

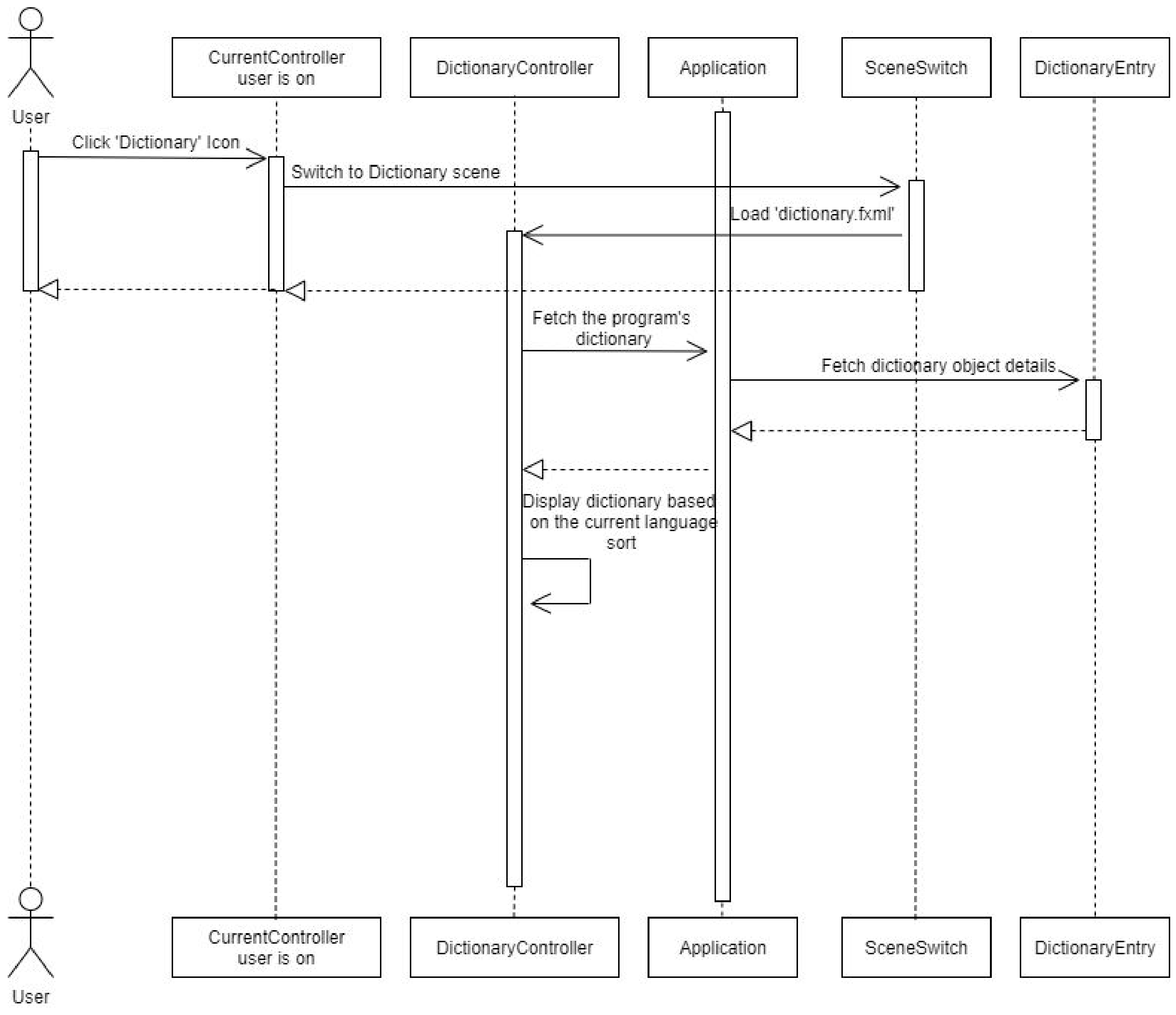
User
CurrentController user is on
DictionaryController
Application
SceneSwitch
DictionaryEntry
Click 'Dictionary' Icon
Switch to Dictionary scene
Load 'dictionary.fxml'
Fetch the program's dictionary
Fetch dictionary object details
Display dictionary based on the current language sort

Figure 2: Sequence diagram for displaying the Dictionary

5.1.2 Use Case 2 Search for a word

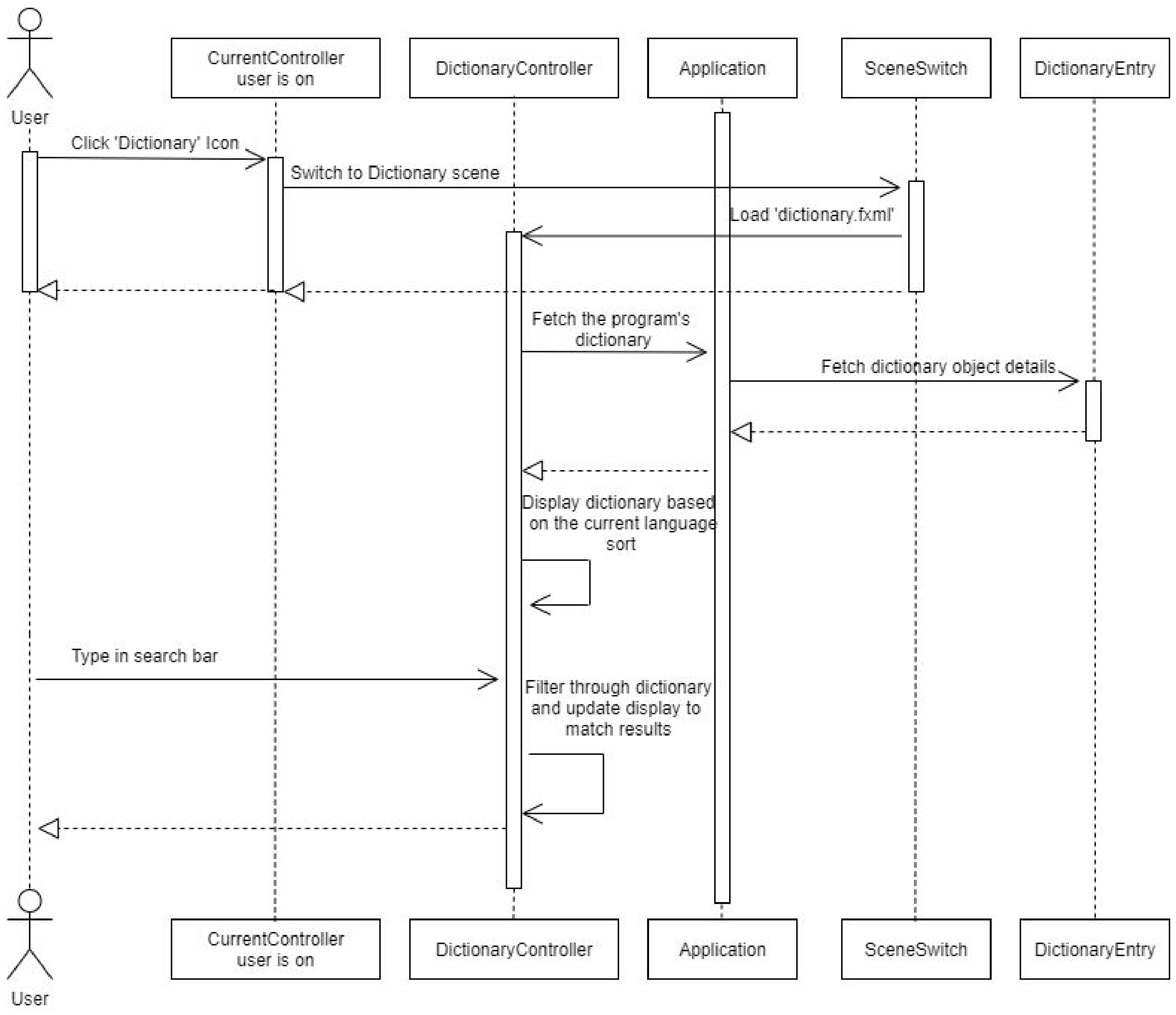
User
CurrentController user is on
DictionaryController
Application
SceneSwitch
DictionaryEntry
Click 'Dictionary' Icon
Switch to Dictionary scene
Load 'dictionary.fxml'
Fetch the program's dictionary
Fetch dictionary object details
Display dictionary based on the current language sort
Type in search bar
Filter through dictionary and update display to match results

Figure 3: Sequence diagram for performing word search on the 'Dictionary' page

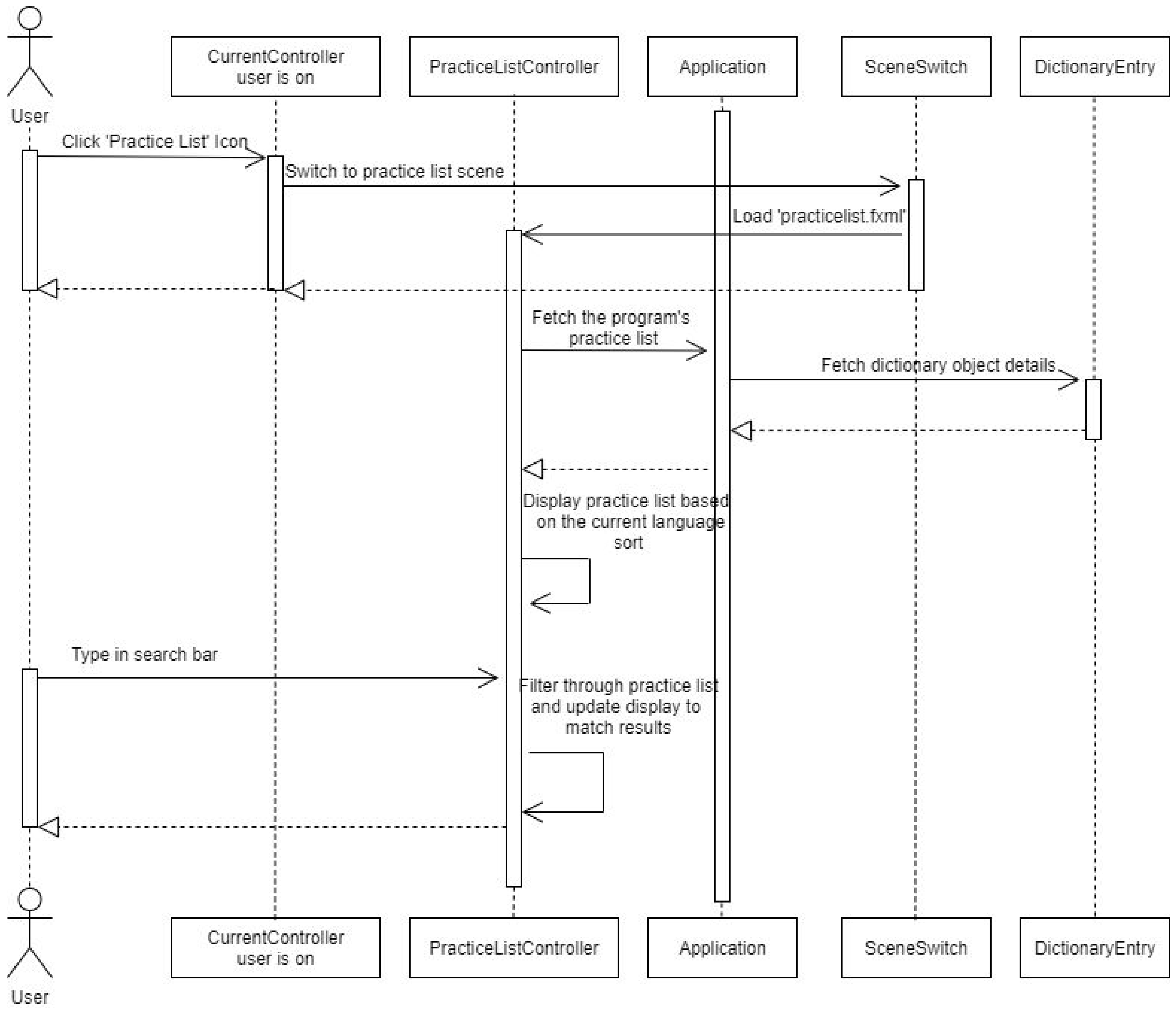
User
CurrentController user is on
PracticeListController
Application
SceneSwitch
DictionaryEntry
Click 'Practice List' Icon
Switch to practice list scene
Load 'practicelist.fxml'
Fetch the program's practice list
Fetch dictionary object details
Display practice list based on the current language sort
Type in search bar
Filter through practice list and update display to match results

Figure 4: Sequence diagram for performing word search on the 'Practice List' page

5.1.3 Use Case 3 View practice list

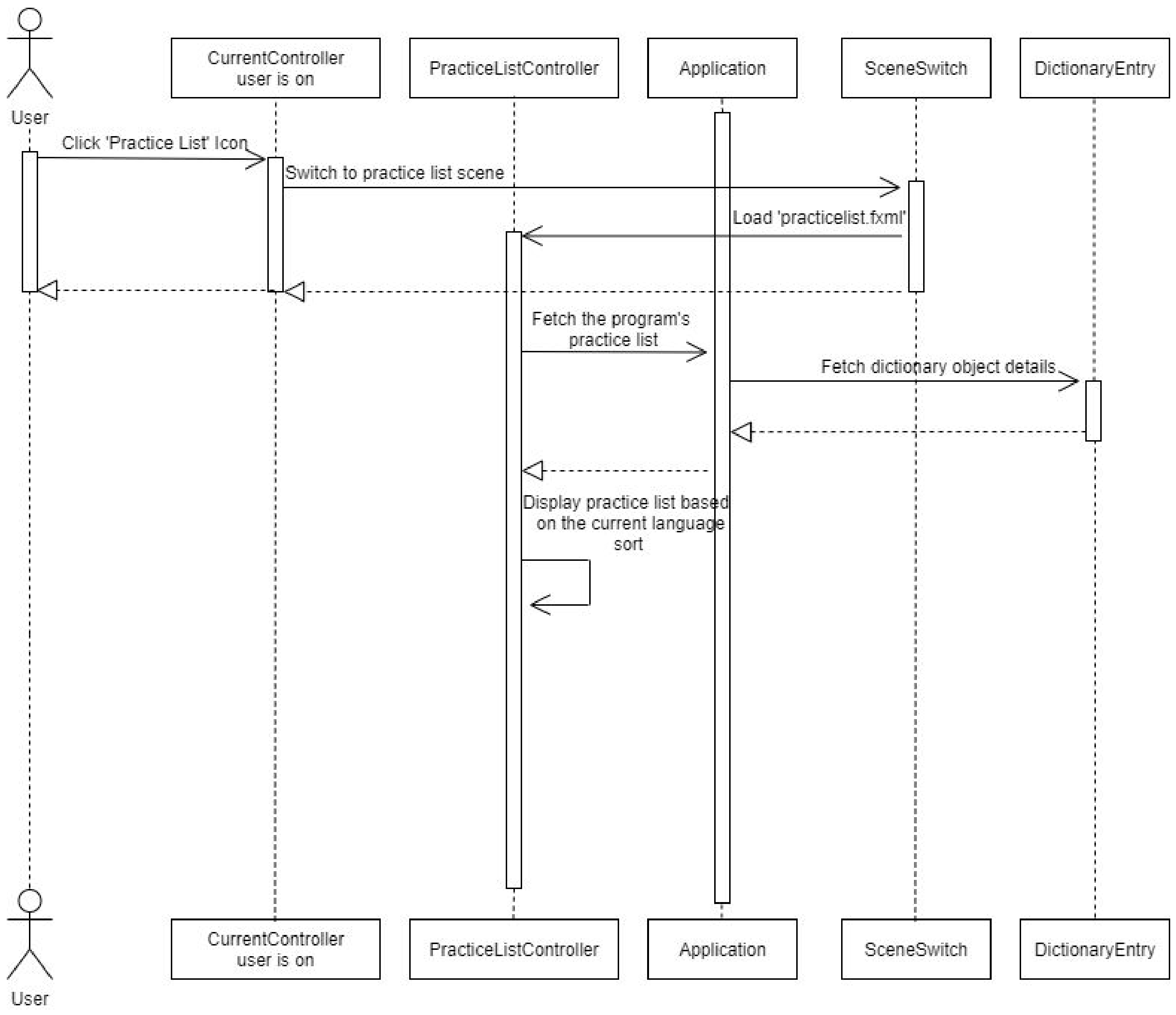
User
CurrentController user is on
PracticeListController
Application
SceneSwitch
DictionaryEntry
Click 'Practice List' Icon
Switch to practice list scene
Load 'practicelist.fxml'
Fetch the program's practice list
Fetch dictionary object details
Display practice list based on the current language sort
User
CurrentController user is on
PracticeListController
Application
SceneSwitch
DictionaryEntry

Figure 5: Sequence diagram for displaying the practice list

5.1.4 Use Case 4 Modify the practice list

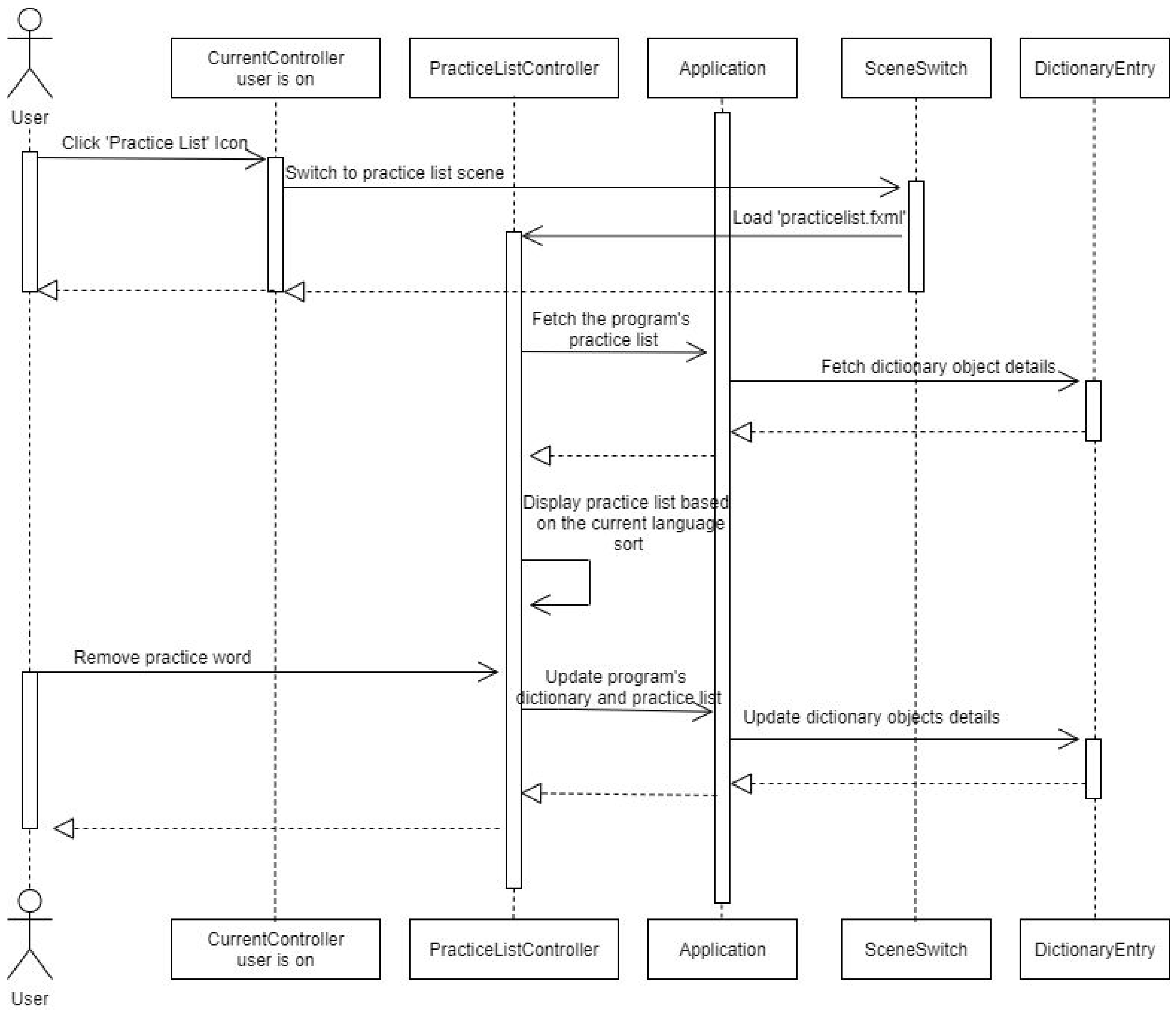
User
CurrentController user is on
PracticeListController
Application
SceneSwitch
DictionaryEntry
Click 'Practice List' Icon
Switch to practice list scene
Load 'practicelist.fxml'
Fetch the program's practice list
Fetch dictionary object details
Display practice list based on the current language sort
Remove practice word
Update program's dictionary and practice list
Update dictionary objects details

Figure 6: Sequence diagram for removing words from the practice list

5.1.5 Use Case 5.1 Start ‘Match The Meaning’ test

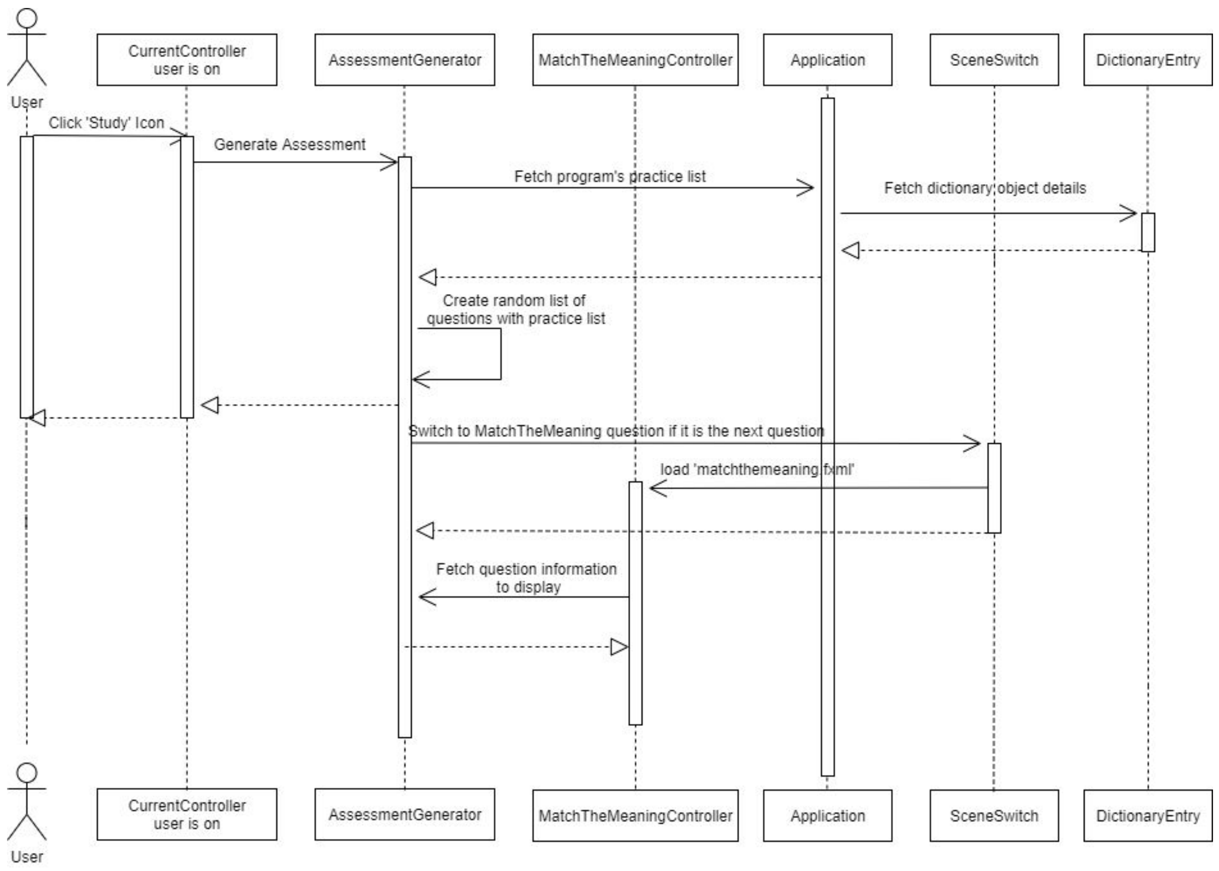
User
CurrentController user is on
AssessmentGenerator
MatchTheMeaningController
Application
SceneSwitch
DictionaryEntry
Click 'Study' Icon
Generate Assessment
Fetch program's practice list
Fetch dictionary object details
Create random list of questions with practice list
Switch to MatchTheMeaning question if it is the next question
load 'matchthemeaning.fxml'
Fetch question information to display

Figure 7: Sequence diagram for the ’Match The Meaning’ test

5.1.6 Use Case 5.2 Start '6 Meanings' test

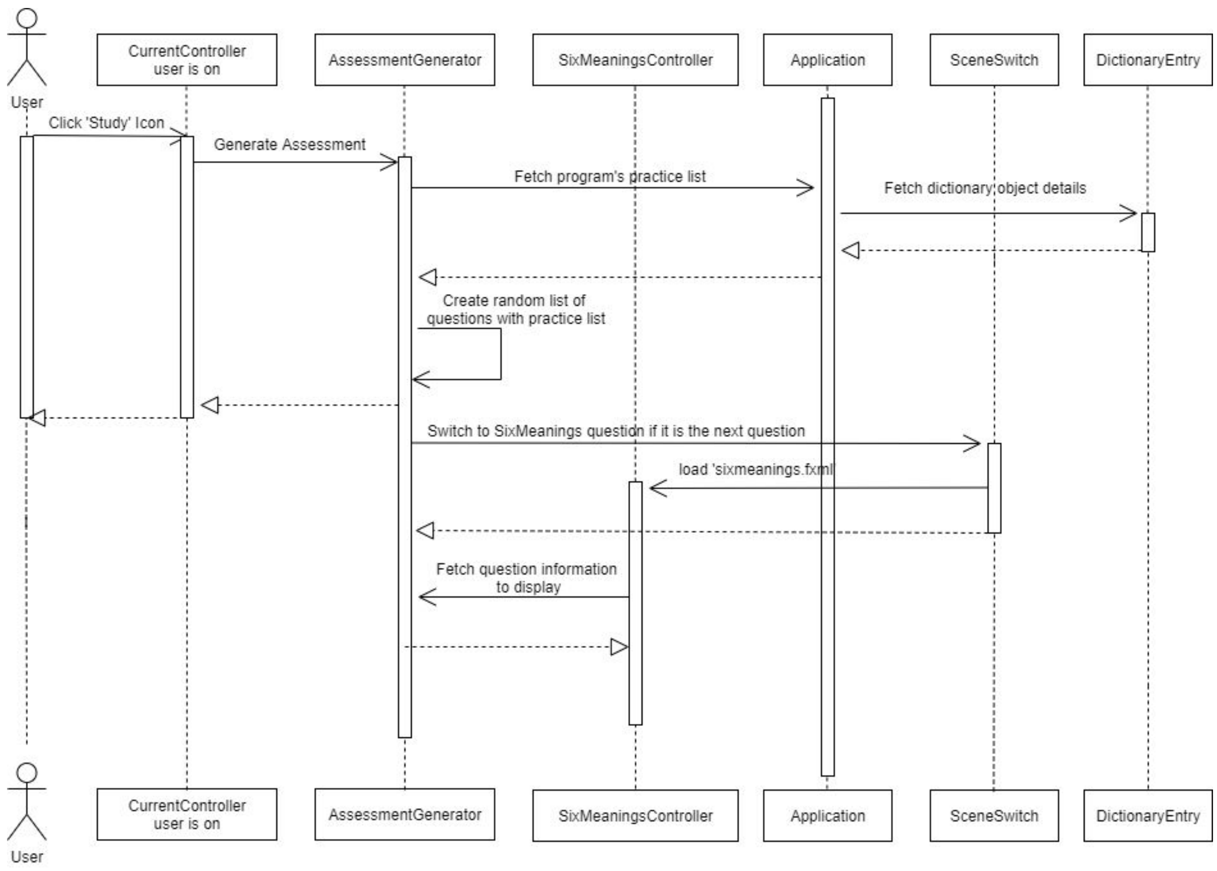
User
CurrentController user is on
AssessmentGenerator
SixMeaningsController
Application
SceneSwitch
DictionaryEntry
Click 'Study' Icon
Generate Assessment
Fetch program's practice list
Fetch dictionary object details
Create random list of questions with practice list
Switch to SixMeanings question if it is the next question
load 'sixmeanings.fxml'
Fetch question information to display

Figure 8: Sequence diagram for the '6 Meanings' test

5.1.7 Use Case 5.3 Start 'Translation' test

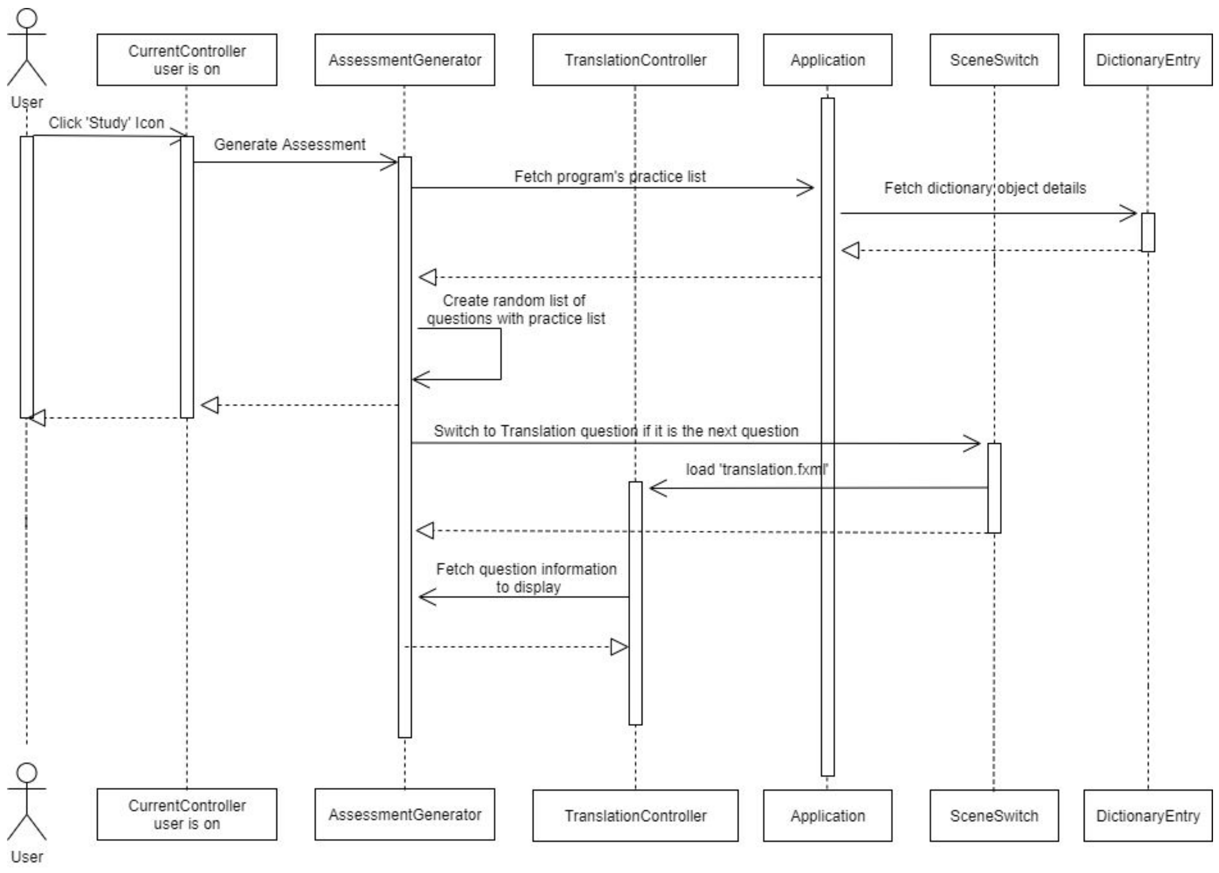
User
CurrentController user is on
AssessmentGenerator
TranslationController
Application
SceneSwitch
DictionaryEntry
Click 'Study' Icon
Generate Assessment
Fetch program's practice list
Fetch dictionary object details
Create random list of questions with practice list
Switch to Translation question if it is the next question
load 'translation.fxml'
Fetch question information to display

Figure 9: Sequence diagram for the 'Translation' test

5.1.8 Use Case 6 View flashcards

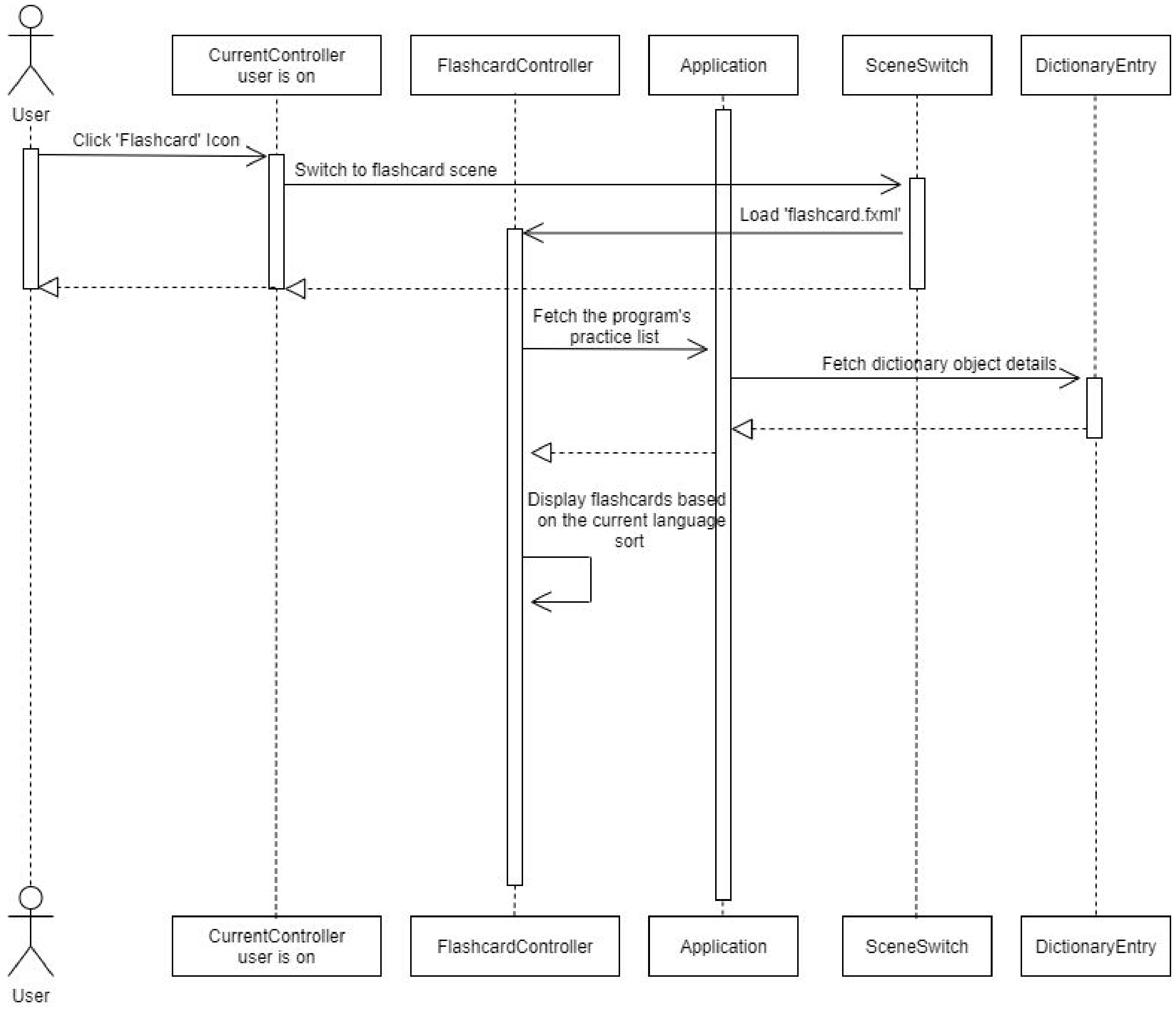
User
CurrentController user is on
FlashcardController
Application
SceneSwitch
DictionaryEntry
Click 'Flashcard' Icon
Switch to flashcard scene
Load 'flashcard.fxml'
Fetch the program's practice list
Fetch dictionary object details
Display flashcards based on the current language sort

Figure 10: Sequence diagram for loading the dictionary list

5.1.9 Use Case 7 Add a new word

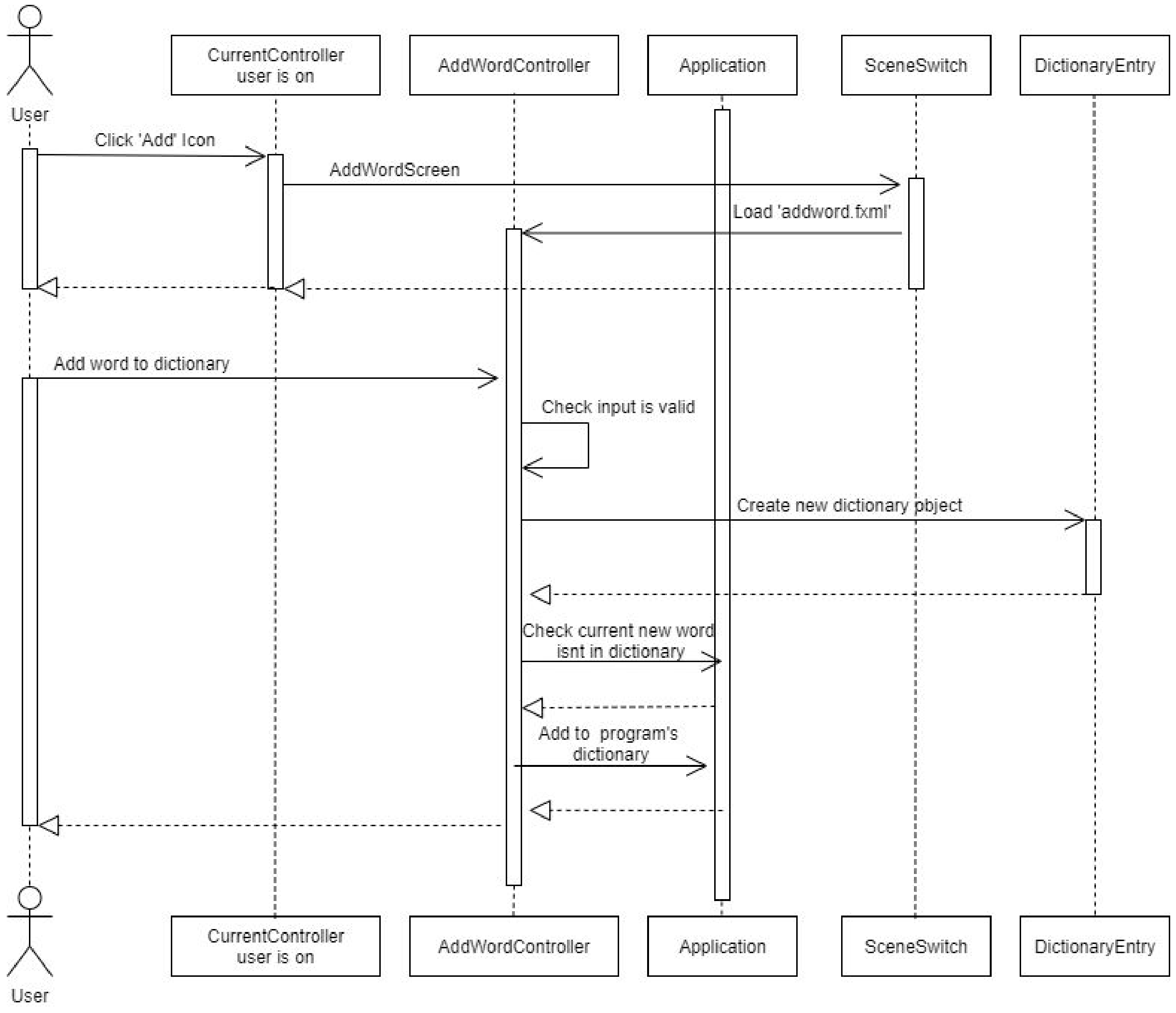

Figure 11: Sequence diagram for adding new words

5.1.10 Use Case 8 Change word ordering

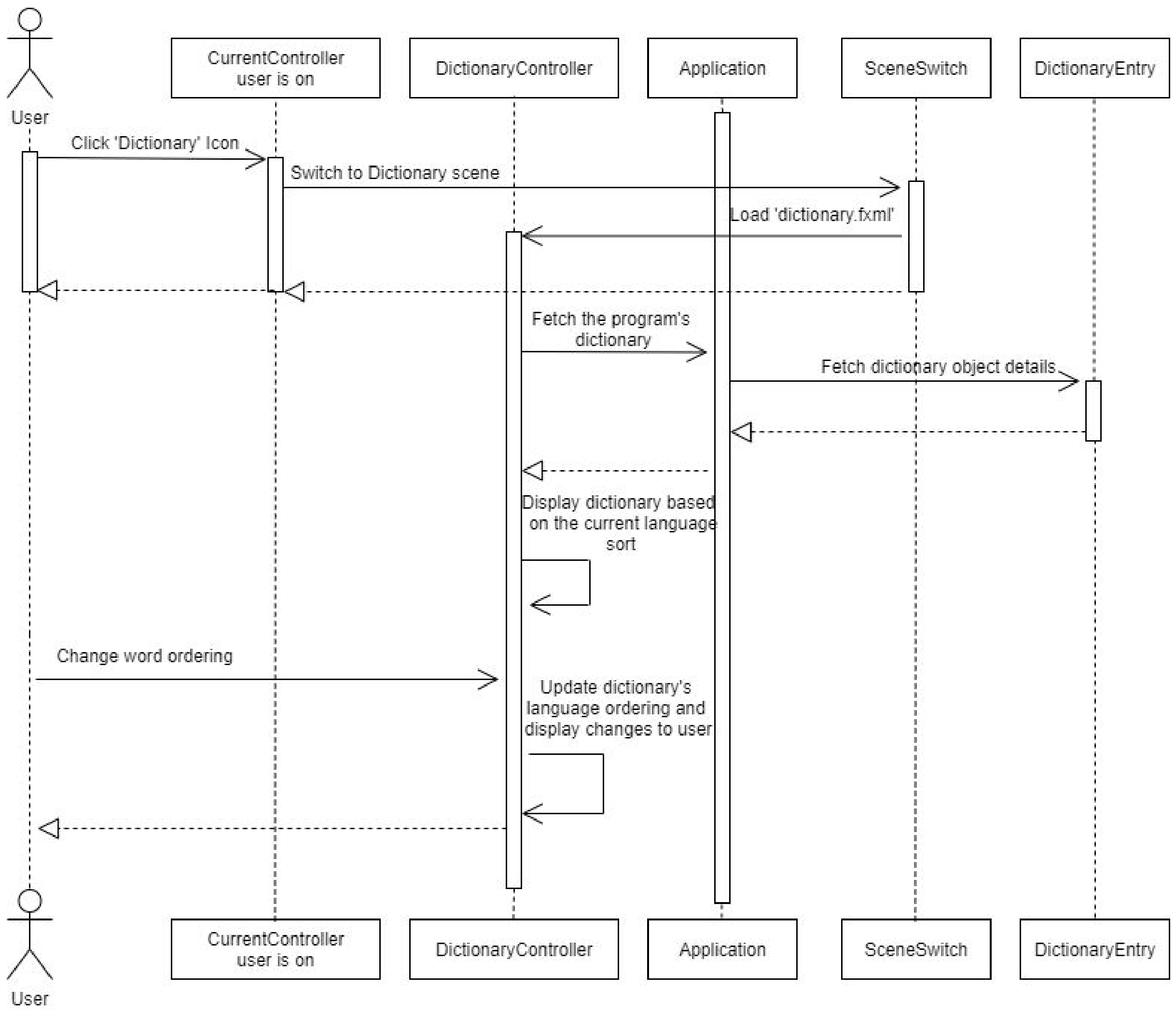
User
CurrentController user is on
DictionaryController
Application
SceneSwitch
DictionaryEntry
Click 'Dictionary' Icon
Switch to Dictionary scene
Load 'dictionary.fxml'
Fetch the program's dictionary
Fetch dictionary object details
Display dictionary based on the current language sort
Change word ordering
Update dictionary's language ordering and display changes to user

Figure 12: 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 ScreenSwitch 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 algorithm uses a lambda expression with a listener to update the filter on a filtered list everytime the textbox which the user types their query into updates. This allows the system to search and filter the list as the user is typing.

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. The new object is checked to see if it is already in the program’s ‘dictionary’ in the Application class. If the new definition doesn't already exist, it is then added to the ‘dictionary’ and ‘practice List’ lists in Application, which will be 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 system start, in the Main method of Application. It will use JavaFX to open a fileChooser, 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 Application’s ‘dictionary’ list with any practice words also being added to the ‘practiceList’ list in Application.

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. 1.4 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
1.6	58	04/05/2020	Made changes to bring compliance with Standard Specifications and Issue Ticket	LW, TP, KB
1.7	58	05/05/2020	Made minor changes to correct missed issues as stated in issue #58	LW, TP, KB