**Computer Science Internal Assessment: French Vocabulary Memorization Software**

CRITERION B – DESIGN OVERVIEW:

# **Use of Flutter and Dart:**

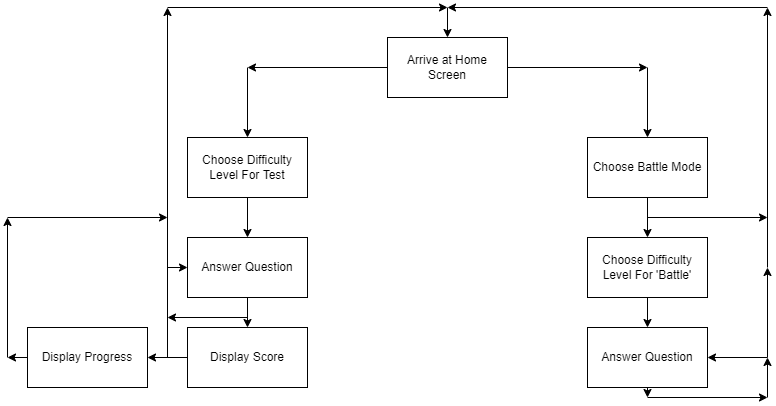
The following features of Flutter and Dart were used to develop the French Vocabulary Memorization Software:

1. Flutter is an open-source UI Software Development Kit for mobile applications that is developed by Google. Dart is the programming language that Flutter was built with and is used with.
2. The Dart Scaffold class has been used to create a basic visual layout for each page. Scaffolds enable the display of Material Design (a design language by Google) elements on screen.
3. Dart Lists have been used to store multiple values in one variable. In the code for the software, I employed only growable lists containing string values, as opposed to fixed-length lists.
4. Dart Operators and Operands have been used throughout the software’s code: Arithmetic operators were used to perform numerical calculations, while relation, assignment, and logical operators were used in various comparisons involving variables.
5. Dart’s Map data structure--a collection of ‘key-value’ pairs--was used to create a routing system that directs the user between the software’s different screens.
6. Dart loops like for and do…while have also been used. These loops were used to execute a block code either a set number of times, or until a certain condition was fulfilled.
7. Dart decision making constructs, specifically if…else statements, have been used to execute a code snippet depending on whether a certain condition is satisfied.
8. Dart’s TextEditingController class was utilized to handle user input. This controller tracks changes in the text inside a text field. The text can then be extracted from the text field by assigning the controller to another variable.
9. Dart’s setState method was used in the software code to rebuild the states of objects and update the values of variables that affected the user interface’s appearance.
10. Dart’s Future.delayed constructor was used to create a short time delay between when a user submits their answer to a question, and when the software moves onto the next question in the test.
11. A Flutter plugin named “Path\_Provider '' was used to asynchronously return a commonly accessed directory on the device’s file system. The user’s score on a test was then written to a local text file at this path. After the subsequent test, the file was accessed again to ascertain the user’s progress in score.
12. Flutter’s “pubspec. yaml '' file was used to specify the metadata needed to import the “Oswald” font package as a dependency of the software. This was done to tailor the appearance of the application to the client’s needs, thus improving its usability.

A few alternatives to developing the French Vocabulary Memorization Software as an Android application with Flutter and Dart were considered:

* I considered developing a computer-based solution. However, the client wanted to be able to use the software anywhere, and a computer is less portable than a mobile phone. This made a mobile application more suited to the client’s needs. Furthermore, the client felt more comfortable with a mobile phone’s intuitive touch screen, rather than a computer’s keyboard and mouse. Hence, using the software on a computer would impair its usability for the client.
* I also considered developing a web-application, but eventually decided against it, as this software’s users are limited to only the client. Web-applications are more suited to instances where the software is used by many people, as opposed to just one person.

**Overall Structure:**



# 

# 

**Prototype Input/Output Screens:**

Figure 1


*Note: See Appendix A for the Client’s feedback on the screens.*

**Figure 1:** This is the home screen of the app. From here, the user can take six types of tests, which are each classified into one of two groups: *French to English,* where the user gets given a word in French and they have to give the English equivalent, and *English to French*, which is the exact opposite. Under both two groups, the user can choose between three difficulty levels: easy, medium, and hard. This changes the questions asked in the test. After any of the six buttons are pressed, the user is taken to the relevant test (Figure 2). Alternatively, if the user prefers to compete against the software on a shared test, they can press the ‘Battle Mode!’ button. They are then directed to a screen where they can select the difficulty level of the ‘battle’ (Figure 5), and subsequently start playing.

Graphical user interface, diagram

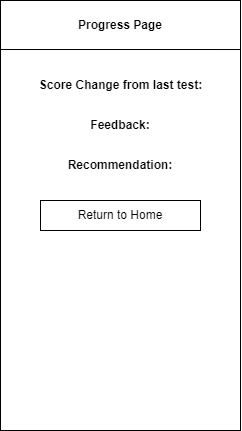
Description automatically generated with medium confidence

**Figure 2:** This is the test interface of the app. Here, the user can take the type of test they chose at the home screen (Figure 1). The user enters their answer to each question in the text box above ‘Answer Status’. On pressing ‘Submit’, a message is displayed below ‘Answer Status’, telling the user if their answer is right or wrong. If the user’s answer is correct, the message will read “Correct!”. If it is incorrect, the message will read “Sorry, that was incorrect. The expected answer was [answer].” After this message is displayed, there is a short time delay, before the user is directed to the next question in the test. Following 10 questions, the user is automatically directed to the ‘Results Page’ (Figure 3), where their score on the test is displayed.

Diagram

Description automatically generated with low confidence

**Figure 3:** This is the results page of the app. Here, the user’s score on the test (out of 10) is displayed. After viewing their score, the user has three options: Firstly, they can press the ‘Take another Test’ button to take another test of the same type (Figure 2). Alternatively, they can press the ‘Return to Home’ button to return to the home screen (Figure 1). Or, lastly, they can press the ‘View Progress’ button to check their progress. On doing so, they will be directed to the ‘Progress Page’ (Figure 4).

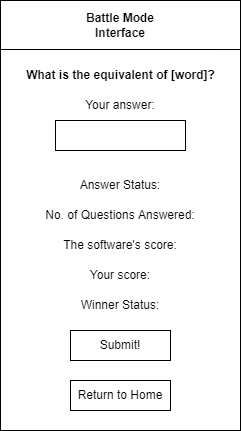


**Figure 4:** This is the progress page of the app. It displays the change in the user’s score from the previous test they took. Below that, the software displays some feedback on the user’s progress, as well as which difficulty level it thinks the user is suited to--based on their latest test score. Through all this information, the user can better assess their current language proficiency. Additionally, the user can return to the home screen by pressing the ‘Return to Home’ button. This helps ensure that the user can navigate to the home screen from any other screen in the software.

Diagram

Description automatically generated

**Figure 5:** In this screen, the user can choose the difficulty level of the software they will be competing against in “Battle Mode” (i.e. how often it will correctly answer questions). After the user has selected one of the Easy, Medium, or Hard difficulty levels, the software will then direct them to the appropriate ‘Battle Mode Interface’ (Figure 6).



**Figure 6:** This is the ‘Battle Mode Interface’. Here, the user answers questions similarly as in the ‘Test Interface’ (Figure 2), the only exception being that both French and English equivalents are asked, as opposed to just one or the other. Additionally, as the user competes against the software, this screen displays information about their ‘battle’ (in descending order): whether the user’s answer to the current question is correct, the total number of questions answered, the software’s score, the user’s score, and, lastly, which one of them is winning. In a battle, the user is free to answer as many questions as they like. Once they finish playing, the user can return to the home screen (Figure 1) by pressing the ‘Return to Home’ button.

**UML Diagram:**

The following is a UML Activity Diagram for my French Vocabulary Memorization Software. It begins with the software loading up the home screen.

Diagram

Description automatically generated

**Logic:**

## *Scripts:*

The logic for the application is divided into various Dart scripts that pass information to each other through import statements. The Dart scripts currently implemented are described below.

Nearly identical scripts are not shown below. For instance, only the test interface script for one test type (a medium-level French to English test) is included, since the test interface scripts for other test types are identical--bar the questions they include, and their associated answers. The same applies to all the “Battle Mode Interface” scripts for varying test types.

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **Name** | **Purpose** |
| 1 | *main.dart* | This script implements a routing system that directs the user between different screens |
| 2 | *home.dart* | The script establishes the home screen of the software |
| 3 | *test.dart* | This script handles the following:   * Generates tests by randomly picking words to ask * Displays each test question to the user * Grades the user’s answers and issues feedback when necessary * Tracks the user’s score on the test |
| 4 | *score.dart* | This script does the following:   * Calculates the change in the user’s score since their previous test * Displays the user’s score on the current test * Determines which difficulty level to recommend to the user * Determines what feedback to give to the user on their progress since the previous test * Displays three options to the user: returning to the home screen, taking another test of the same type, or viewing their progress since the previous test |
| 5 | *progress.dart* | This script handles the following:   * Displays the change in the user’s score since the previous test * Displays the system’s feedback on the user’s progress * Displays the difficulty level recommendation for the user |
| 6 | *battlemode.dart* | This script displays three difficulty levels the user can pick for Battle Mode: Easy, Medium, and Hard |
| 7 | *bmeasy.dart* | This script does the following:   * Simulates a ‘battle’ of the selected difficulty level * Randomly picks words to ask the user and the software * Handles grading and feedback for the user’s and the software’s answers * Tracks and displays various details of the battle: whether the user’s answer to the current question is correct, the total number of questions answered, the software’s score, the user’s score, and, lastly, which one of them is winning. |
| 8 | *F\_O.dart* | This script handles the following:   * Writes the user’s score on each test to a local text file * Retrieves this score after the subsequent test in order to determine the user’s progress |

# **Explanation of Variables by Script:**

Note: Though the test interface scripts for varying test types all use the same variables, their names may change with test type. For instance, the name of the variable *answer* changes to *answerE* in the script for an easy French to English test, and answerEA for an easy English to French test. This applies to the names of variables in Battle Mode scripts too.

1. ***test.dart***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** | **Data Type:** | **Description:** | **Other Information:** |
| noCorrect | Integer | Holds the number of questions the user has answered correctly in the current test |  |
| displayNoCorrect | String | Holds the number of questions the user got correct in the current test | Converted from an integer to a string |
| NC | Integer | Holds the number of questions the user has answered correctly | Value is assigned to variable only after current test finishes |
| counter | Integer | Holds the number of questions the user has answered so far in the current test |  |
| expectedAnswer | String | Holds the correct answer to the current test question |  |
| questions | List | Holds all the questions for a given test type |  |
| answers | List | Holds all the answers for a given test type |  |
| isCorrect | Boolean | Holds a value indicating whether the user’s answer to the current question is correct. |  |
| answerMessage | String | Holds the software’s feedback on the user’s answer to the current question |  |
| noQuestions | Integer | Holds the total number of questions classified under a given test difficulty level |  |
| lastScoreM | Integer | Holds the user’s score on their previous test. |  |
| answer | String | Holds the user’s answer to the current question |  |
| word | String | Holds the word whose equivalent is being asked in the current question |  |
| element | Integer | Holds the index number (in the *questions* and *answers* lists) of the current question-answer pair |  |
| indexS | Set | Holds the index numbers (in the *questions* and *answers* lists) of all the question-answer pairs on the current test | Converted from a Set to a List |
| indexes | List | Holds the index numbers (in the questions and answers lists) of all the question-answer pairs on the current test |  |
| questionList | List | Holds all the questions on the current test |  |
| answersList | List | Holds the answers to all the questions on the current test |  |
| testQ | List | Holds all the questions on the current test | Temporary Variable |
| testA | List | Holds the answers to all the questions on the current test | Temporary Variable |

1. ***score.dart***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** | **Data Type:** | **Description:** | **Other Information:** |
| currentScore | Integer | Holds the user’s score on the current test |  |
| scoreChange | Integer | Holds the change in the user’s score from the previous test to the current test |  |
| displayscoreChange | String | Hold the change in the user’s score from the previous test to the current test | Converted from an integer to a string |
| displayProgressText | String | Holds the software’s feedback on the user’s progress from the previous test |  |
| recommendation | String | Holds the software’s difficulty level recommendation to the user | Based on the score the user obtained in the current test |

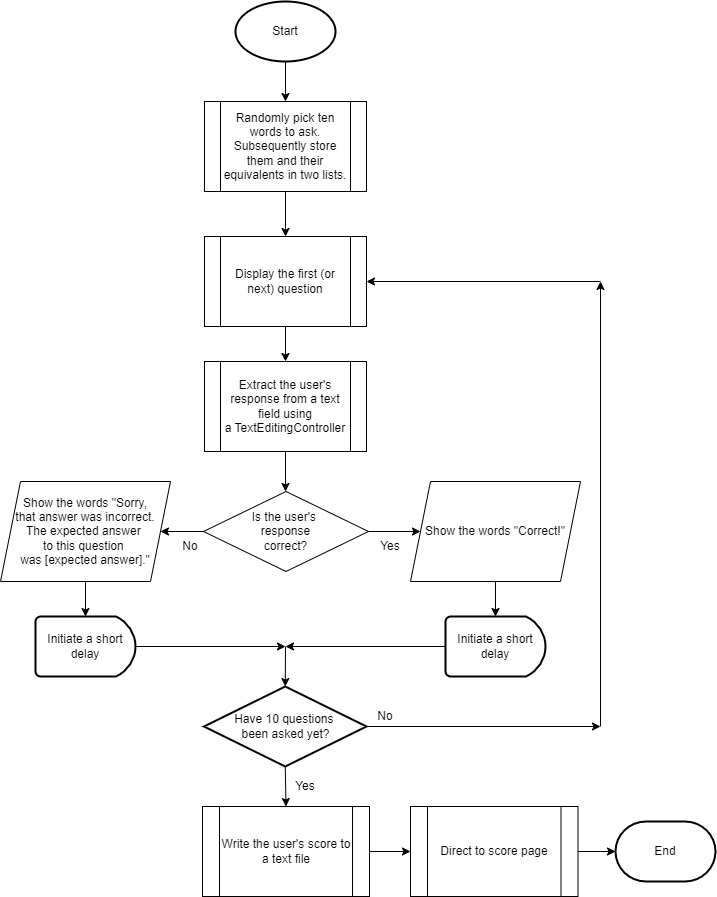
1. ***bmeasy.dart***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** | **Data Type:** | **Description:** | **Other Information:** |
| noCorrect | Integer | Holds the number of questions the user has answered correctly in the battle |  |
| displayNoCorrect | String | Holds the number of questions the user has answered correctly in the battle | Converted from an integer to a string |
| issoftwareCorrect | Boolean | Holds a value indicating whether the software’s answer to the current question is correct |  |
| nosoftwareCorrect | Integer | Holds the number of questions the software has answered correctly |  |
| displaynosoftwareCorrect | String | Holds the number of questions the software has answered correctly | Converted from an integer to a string |
| counter | Integer | Holds the number of questions that have been asked in the battle |  |
| displayNumberQ | String | Holds the number of questions that have been asked in the battle | Converted from an integer to a string |
| accuracyNumber | Integer | Holds a random number that determines whether the software answers the current question correctly |  |
| element | Integer | Holds the index number (in the questions and answers lists) of the current question-answer pair |  |
| previous | Integer | Holds the index number of the previous question in the *questions* list. This is to ensure the same question is not asked consecutively |  |
| expectedAnswer | String | Holds the correct answer to the current question |  |
| howGood | Integer | Holds a number that determines the difficulty level of the battle |  |
| questions | List | Holds all the questions for a given test type |  |
| answers | List | Holds all the answers for a given test type |  |
| isCorrect | Boolean | Holds a value indicating whether the user’s answer to the current question is correct |  |
| answerMessage | String | Holds the software’s feedback on the user’s answer to the current question |  |
| noQuestions | Integer | Holds the total number of questions classified under a given test type |  |
| winnerStatus | String | Holds text describing who is currently winning : the user or the software |  |
| SoftwareAnswer | String | Holds the machine’s answer to the current question |  |
| wrong | Integer | Holds the index number of a wrong answer to the question in the *answers* list. This number is randomly-generated. |  |
| word | String | Holds the word whose equivalent is being asked in the current question |  |
| answer | String | Holds the user’s answer to the current question |  |

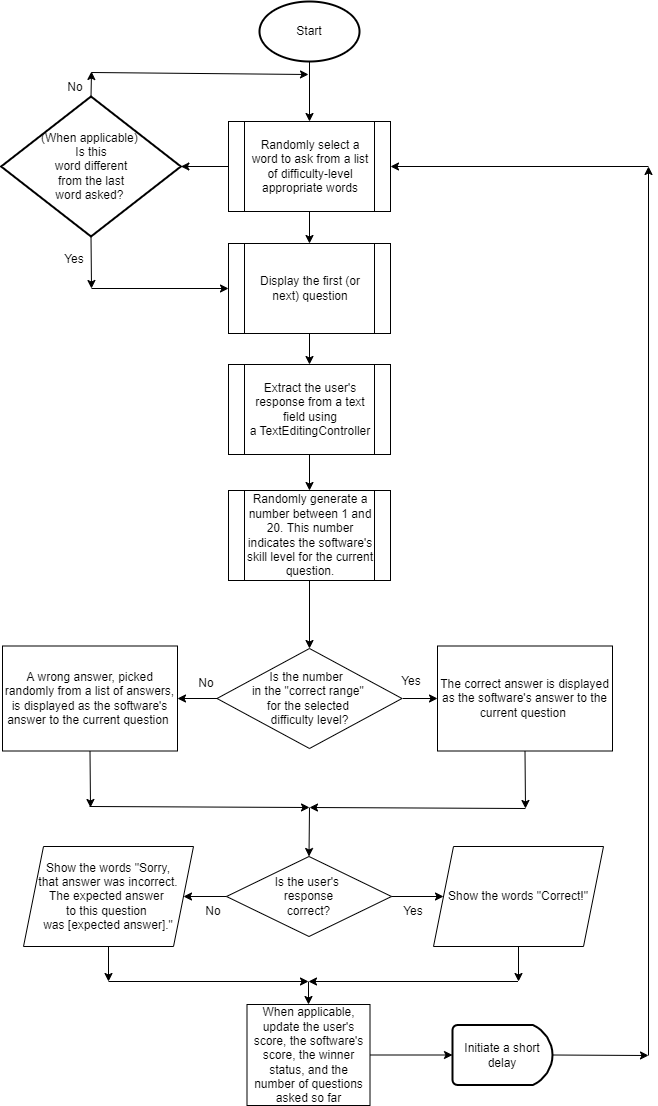
**Key Algorithms:**

There are a few algorithms that are integral to the functioning of the application.

1. **Test Interface:** This flowchart shows the algorithm for randomly generating a 10-question test of the selected type. One by one, questions are displayed, the user’s answer to each one is graded, and appropriate feedback is given:



1. **Battle Mode Interface:** This flowchart shows the algorithm that enables the user to compete against the software on a shared test. One by one, questions of the selected difficulty level are displayed, the user’s answer to each one is graded, and various details of the battle are tracked.



## **Extensibility:**

|  |  |
| --- | --- |
| **Feature** | **Details of extensibility** |
| **Progress** | * The software can record the user’s score on each test, and then plot a line graph showing the user’s score versus time. * The graph can be generated by adding the Flutter “Graphic” Package into the software’s *pubspec.yaml* file |
| **Difficulty Level Recommendations** | * Two factors can be considered while recommending a difficulty level: the user’s score on the current test, and the change in the user’s score from the previous test--as opposed to only the former. |
| **Adding more types of tests** | * In the base project files, an existing test interface script can be duplicated * Then, the script’s question and answer lists can be edited to tailor them to the new test type. * Finally, to link this new script to the other Dart scripts, it should be referenced in the ‘main.dart’ file |

## 

## **Test Plan:**

| **Test Action** | **Testing Method** | **Successful Outcome** | **Success Criteria** |
| --- | --- | --- | --- |
| Home Screen | Loading up the application | The home screen has six buttons, each one for a different type of test. On pressing a button, the user is directed to the test interface of the selected test type. | 1 |
| Test Generation | On the home screen, press each button representing each type of test. Then, solve each of the corresponding tests in their entirety. | As soon as the test interface loads, the first question in the test is displayed. After each question, a new, randomly selected question is displayed. In total, 10 questions are asked before a test ends. | 2 |
| User Input | Click on the empty text field, and type using the keyboard, and press the submit button | The user can type in the text field and press the submit button. | 3 |
| Feedback on Correct Answers | Input the correct answer to a question and press the submit button | A message that reads “Correct!” is displayed on screen. This stays on screen for a few seconds. Then the software displays the next question. | 4 |
| Feedback on Incorrect Answers | Input a wrong answer to a question and press the submit button | A message that reads “Sorry, that was incorrect. The expected answer was [expected answer].” is displayed on screen. This stays on screen for a few seconds. Then the software displays the next question. | 5 |
| Score Screen | Solve six tests in their entirety, one for each type of test | After the end of each test, the user is automatically redirected to a page where they can see their score (out of 10) displayed. This number should be correct. | 6 |
| View Progress | For each type of test, press the ‘View Progress’ button on the Score Page. | The user is immediately directed to the Progress Page. Here, the change in the user’s score from the previous test is displayed. This number should be correct. Additionally, below the number, some text describing the score change, (e.g., “Your score has improved since the previous test”) is displayed. | 7 |
| Difficulty Level Recommendations | For each type of test, press the ‘View Progress’ button on the Score Page. | The progress page displays the recommendation below the score change. It should be one one of three difficulty levels: easy, medium, or hard. Furthermore, this recommendation changes with large variations in the user’s test score. | 8 |
| Difficulty Levels in Battle Mode | On the home screen, press the “Battle Mode” button. | The user is directed to a page where they can choose between three difficulty levels for the battle: Easy, Medium, and Hard. | 9 |
| Battle Mode | On the home screen, press the “Battle Mode” button. On the subsequent screen, press each button corresponding to each difficulty level of Battle Mode. Then, play Battle Mode for each of the difficulty levels. | As soon as the Battle Mode interface loads, the first question is displayed. After each question, a new, difficulty-level appropriate question is displayed. The user can also enter their answers to each question, and duly receive feedback on them. Lastly, various details of the battle are tracked and displayed on screen: whether the user’s answer to the current question is correct, the total number of questions answered, the software’s answer to the current question, the software’s score, the user’s score, and, lastly, which one of them is winning. | 10 |
| Navigate to the Home Screen | Attempt to return to the home page from any screen | The home screen of the software is displayed | 11 |

**Note:** User acceptance testing will be done after the software is fully developed.