Program Manual

Table of Contents

Part 1: Introduction

The Ducktionary is a dictionary application made in Java for PC. Dictionary applications that are on PC today don't really look that neat with outdated UI. Therefore to combat that,this dictionary application for PC has Material Design in mind. So that compatibility isn't an issue, the application is created with the Java programming language. Therefore this should work on most PCs since Java is usually installed in most PCs already. However, since we will be creating the dictionary app with Material design in mind, Java itself cannot handle the complexities of material design. This is why other than java, JavaFX and especially the JFoenix library which is an external library that allows the usage of material design in a Java application is used.

Just like other dictionaries when the user searches for a word the following appears:

1. Parts of Speech :

   (Verb/Noun/Pronoun/Adjective/Adverb/Preposition/Conjunction/Interjection etc.)

2. The word itself
3. Definition (When applicable, there will be multiple definitions)

Other Notable Features:

1. Splash Screen
2. Welcome Onboarding with different background colors and animations
3. User account login/logout/register
4. User save favorite words
5. User tracking: search history, added word history, removed word history, and store user information.
6. Keyboard shortcuts
7. Tooltip hints on everything when the cursor is hovered
8. Cursor variations on buttons and interactive entities.
9. Animations
10. Custom Search/Index
11. Button feedback when clicked
12. Add/Remove words
13. Top quick search bar
14. User dashboard
15. Comprehensive help screen with FAQs and detailed video tutorials
16. Word not found page with animation
17. Offline/Online database with SQL

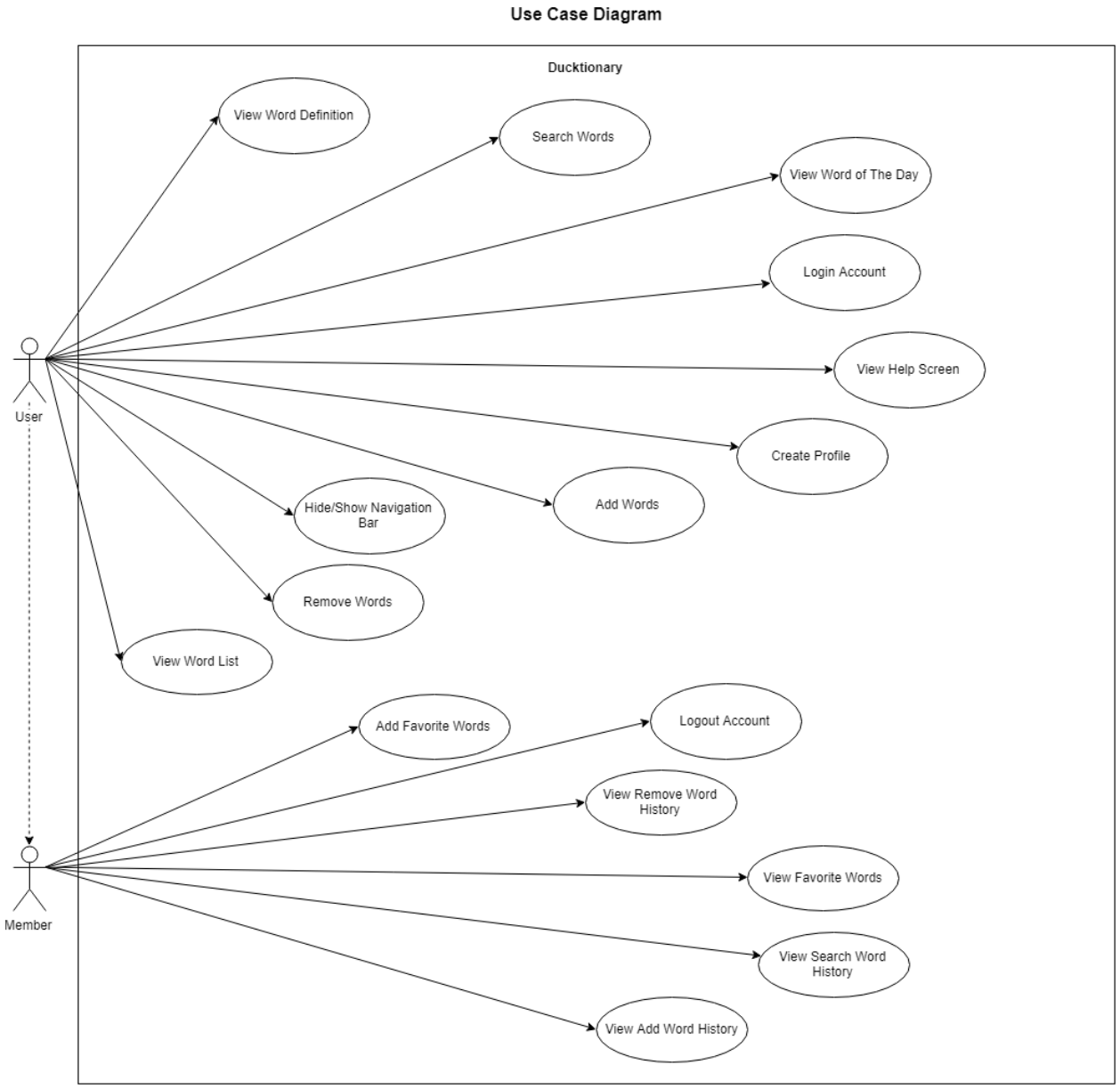
18. External database using ".txt" file with input/output functions
19. All button designs
20. Different background colors for certain windows
21. Over than 20+ icons and icon designs
22. Navigation bar/menu bar
23. All forms captioned
24. Click the top logo to go to dashboard

Part 2: Getting Started

- **System requirements**
    - Windows 10 PC
    - Java 8 and JavaFX 8- https://www.java.com/en/download/
    - Recommended IDEs (choose one):
        - e(fx)clipse - https://www.eclipse.org/efxclipse/index.html
        - intelliJ - https://www.jetbrains.com/idea/
- **Configuring the Source Code**
    - Open the source code folder
    - Include the following libraries on your project
        - ControlsFX8 - http://fxexperience.com/controlsfx/
        - JFoenix8 - http://www.jfoenix.com/
    - Configure your IDE to use UTF-8 settings for the project
    - Run the Main.java file to test the application.

Part 3: Developer's Guide

Use Case Diagram



Description of Features

Dictionary Contents

Considering the size of our team, our budget, and our schedule currently the contents of the dictionary consist of only English words. Other than that, we have limited our dictionary data

to only store the parts of speech (verb/noun/pronoun etc), the word itself, the definition (if there are multiple definitions, then store only 1 or 2 or 3 definitions).

Database

The dictionary currently has two options to load words in to the database. The first option and the default one is to use the included ".txt" file located in the program folder to load the definitions into the dictionary. The definitions that are inside this dictionary can be modified by adding or removing the words by the user as long as they use the application properly. The second option is to use a local server to load the database of the dictionary form. However to use this option the user must have a local server which hosts the definitions. Since we suspect that not all users will have this feature, for now the current default option is to use the txt file.

User Interface

In terms of UI design JavaFX and ControlsFX are used to supplement the default JavaFX version 8 user interface capabilities. JFoenix and ControlsFX are additional library JavaFX that's designed to better support material design for applications. However, we are definitely limited to what JavaFX, ControlsFX, and JFoenix is able to handle. This is why some sections of the user interface might seem heavy in terms of memory because there are no way to combat the memory usage for the goal that we want to provide. We built the user interface using FXML.

Profile

The application has a profile feature where the user can login to his/her account to see the search history, add a favorite word and view it in the favorite word list, and track the add/remove word history. For now the profile information is saved in a plaintext file called "Profile.txt" on the program folder. For every different variable, each variable is seperated by a "%".

Edit

Editing the dictionary is one of the core functionality of this application where the user can add and delete words from the dictionary. The Edit feature is done by loading the database which if using a local option loads it from the "NewDictionary.txt" file on the user folder. Then when the application is running every time the user edits the data whether it is removing or adding a word the dictionary is refreshed by saving and replacing the txt file with a newer

version that has the newer version of the dictionary loaded from the memory into the txt file which will be appended and sorted.

The txt file can be read like this: First the text file will be searched, if any of the files matches NewDictionary.txt then that file location will be saved. Using that file location the function a class that will contain the words will use it and use the function readTextFile with the parameter of the file with that file location. The function readTextFile takes the text file, use bufferedReader and add each line in the text to an arraylist, this arraylist will contain the word and information about it.

Application Structure and Class Diagram

Since we are using Java and JavaFX for this application, we faced a huge problem in development. It turns out that JavaFX itself is limited in ways of communicating data through different stages/scenes. What happened is that multiple scenes and stages can't communicate with each other directly to send information or do a method/function and then save the variable. Because of that we had to devise a way in which different those different stages/scenes can communicate with each other. Therefore we came up with the solution of creating an intermediary between each scene/stage so that data can be transferred with each other and also share variables and functions. So now multiple different stages/scenes can communicate with each other easily and there is a centralized storage of variables and information. This class is the “DuckHomeController.java” class which is usually referred to in the code as the “mother” class.

The class diagram then looks like below (due to the limitations of this document file, the class diagram image that is more detailed, zoomable, and larger can be found in the same folder as the program manual in the application folder). Notice how each class has to first contact the mother class which is the “DuckHomeController.java” class (the one in the middle of the image where all the lines lead to) before it can communicate to the other classes.

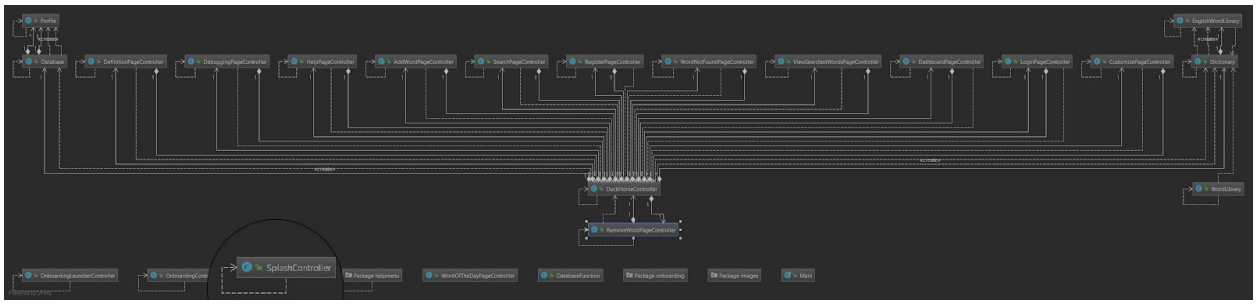
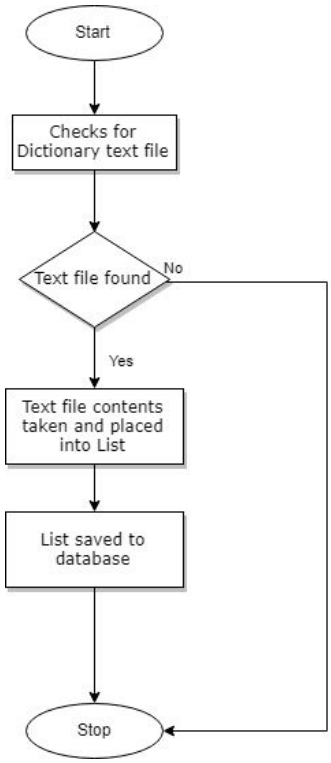


Image of class diagram. To view a larger version open the folder where the Program Manual is located on the Ducktionary install directory.

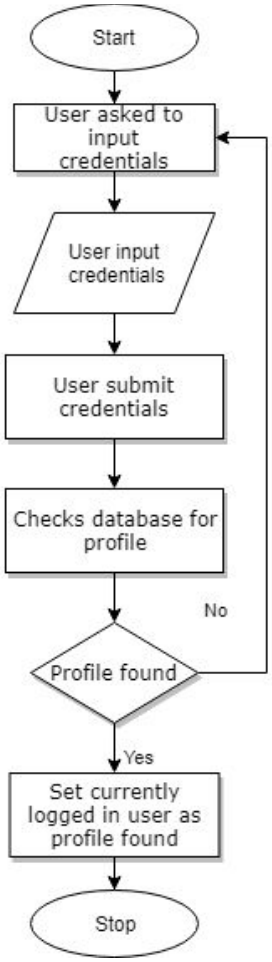
Flowchart of Important Functionalities

Load Dictionary Flowchart



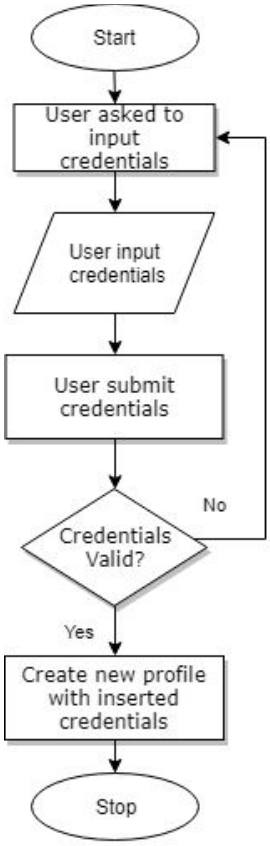
This flowchart loads the dictionary list, it finds for the dictionary text file if its found the contents of the text file is taken. Each line will be inserted as an item in an ArrayList.

Login Flowchart



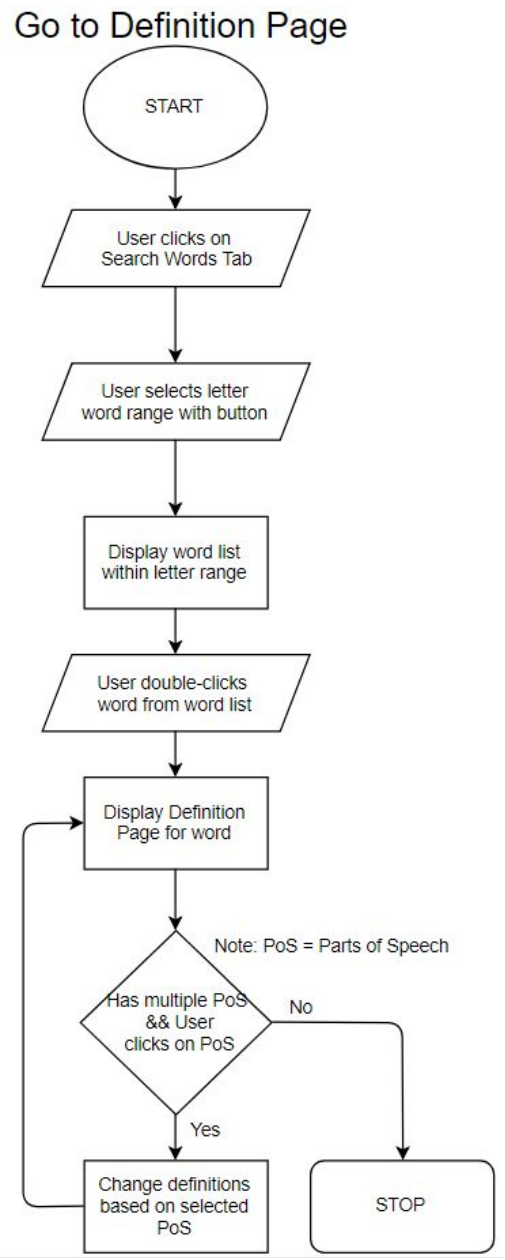
This flowchart shows the process of which a user logs in to an account. First the user is asked for profile credentials that is username and password. As user input the credentials the database is checked if any of the profiles matches the credentials. When found that profile is used as the currently logged in user.

Register Flowchart



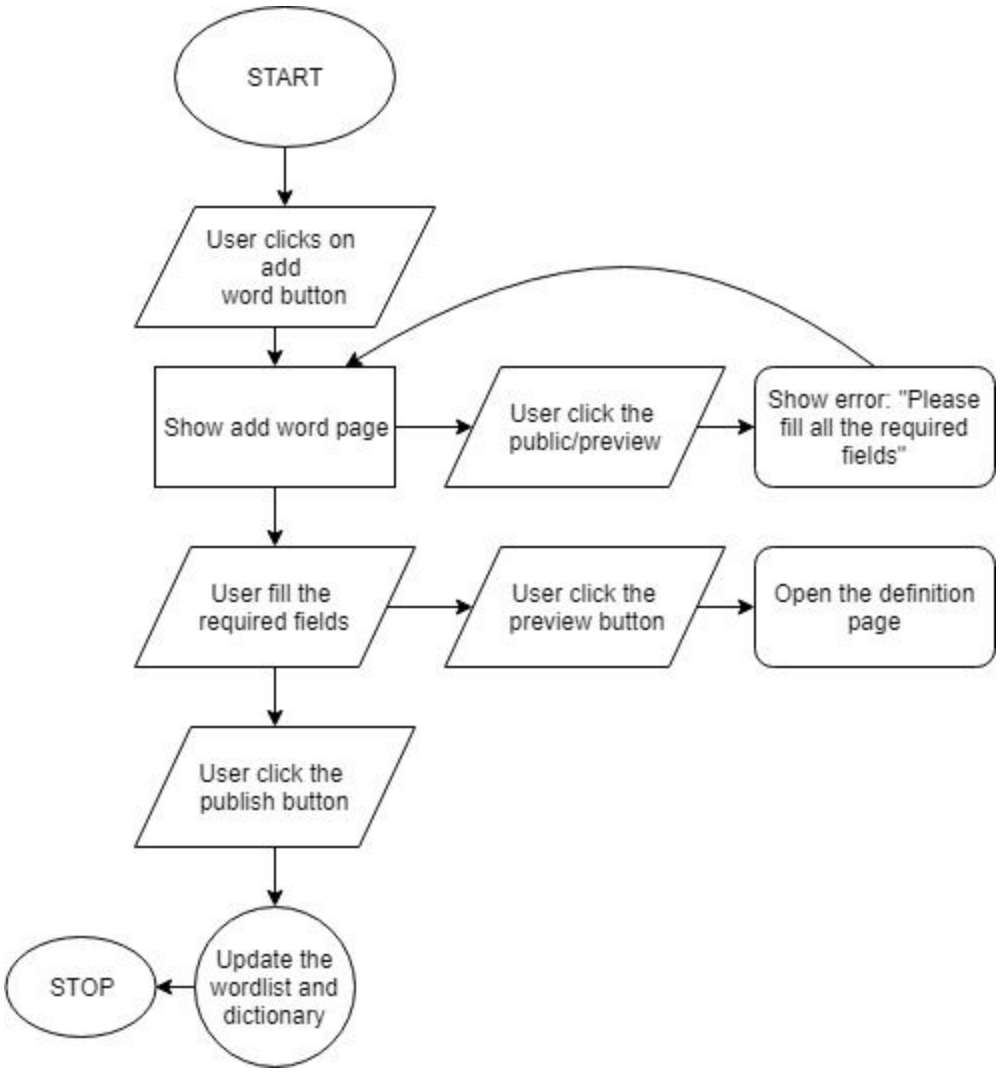
This flowchart shows the process of which a user register an account. The user is asked for credentials that is email, username, password and submits it. If the credentials are valid then a new profile is created.

Go To Definition Flowchart



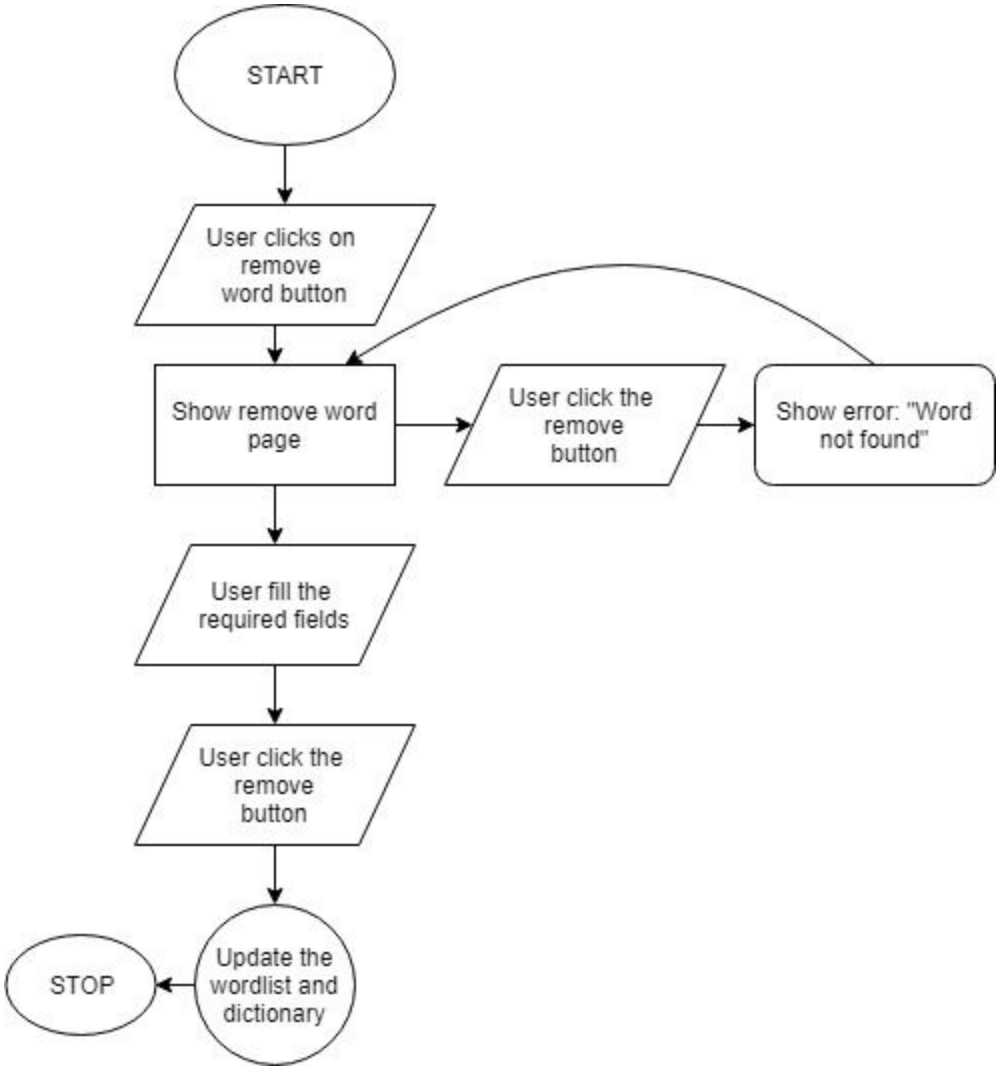
This flowchart illustrates navigating to the search page to find the definition of a particular word. They have a predetermined set of letter ranges to choose from (A-C, D-F, and so on), and selecting one takes them to a new page with the list of words within that letter range. Double-clicking on a word takes them to its definition page, of which there can be 2 sets if there are two parts of speech. Clicking on a part of speech changes the displayed definitions to the ones corresponding to the selected part of speech, with the selected one highlighted in green.

Add word Flowchart



The flowchart shows the process of adding new words. First the addword button redirect the user to add word page. Then it is required to fill all the required fields before clicking the publish button to update the database word .

Remove word Flowchart



The flowchart shows the process of removing existing words. First the removeword button redirect the user to remove word page. Then it is required to fill all the word that wants to be removed on the empty fields and click the remove button to remove the selected word from the database

Classes and Its Methods and Variables

Below are the list of notable functions and variables and its descriptions. Some of the methods or variables will be omitted due to it being trivial (ex: as setter/getter).

AddWordPageController.java (JavaFX layout: AddWordPage.fxml):

This class handles adding a new word into the dictionary, with 4 forms for each variable.

Important Functions:

Function Name	**Description**
public void init(DuckHomeController duckHomeController)	Sets the DuckHomeController for this class, in order to access the home's variables and functions from the Definition Page.
public void previewGo(ActionEvent event)	This function will display the preview of the inputted word in Definition Page.
public void cancelButtonGo(ActionEvent event)	This function will cancel Add Word function and redirects back to Customize Page.
Public void publishGo(ActionEvent event)	This function will add inputted word to dictionary arraylist.

CustomizePageController.java (JavaFX: CustomizePage.fxml):

This class handles the customize page, which allows the user to add or remove words in the dictionary.

Important Functions:

Function Name	Description
void setCustomizePageList()	This updates the list of words that are displayed, depending on the search form in the page. An empty bar will display all the words in the dictionary.
void init()	This function is ran when the program is first started. Unlike some other classes, which only use this function to connect the class with the DuckHomeController, this also sets up the value of the columns in the TableView, and setting a click listener for the rows in tableview that sends the program into the DefinitionPage upon double-clicking a word. It also inserts the dictionary (in a viewable form) into the TableView as well.

Important Variables:

FXML Variable	Description
private DuckHomeController mother	This connects the CustomizePageController class with the main panel (the DuckHomeController class), allowing access to its functions and variables.
public TableView<ArrayList<String>> table	This TableView is where the current list of words (depending on what is searched) is displayed. Each of its TableColumn classes use one value from the ArrayList<String> within this TableView for each row, filling each column with a String value.

DashboardPageController.java (JavaFX layout: DashboardPage.fxml):

This class handles dashboard page, which shows the users information about the user (Ex: search history, favorite words, etc). The page provide navigations to log in, log out, add word, etc.

Important Functions :

Function name	Description
void setLogoutButtonEnabled	Activate the logout button or deactivate it according to if a user is logged in
Void wordOfTheDayGo	Creates a new tab with the word of the day content. Showing the user word of the day
Void addWordsGo	Tells the mother controller to do function triggerAddWordButton when the button that has this function is pressed.
Void getStarted	Tells the mother controller to do function putCursorOnSearchBar when the button that has this function is pressed.
Void login	Tells the mother controller to switch tab to login screen when the button that has this function is pressed.
void logout	Logs the user out, and update changes made to the previously logged in user
void openProfileGo	Change scene to quick tutorial (Method name wasnt changed)
void removeWordsGo	Tells the mother controller to switch tab to

	customize page
void viewHelpGo	Tells the mother controller to switch tab to help page
public void initialize	Currently this only deactivates the logout button
public void init	Sets the mother controller to the duck home controller
Public Label getUsername	Returns the username variable

Important Variables:

FXML Variable	**Description**
private DuckHomeController mother	This connects the CustomizePageController class with the main panel (the DuckHomeController class), allowing access to its functions and variables.
Public ListView historyList	This is where the user's search history will be showed on the screen. The contents will be taken from the currently logged in user's information
Public ListView atAGlanceList	This is where the user's stats will be showed. Contents will also be taken from currently logged in user's information
Public ListView favoriteList	This is where the users favorited words will be showed according to currently logged in user's information
Private TextArea username	This text area will contain the message of who is currently logged in
private JFXButton dashLoginButton	Navigation button to lead user to the login page
public Label titleLabel	The title
public JFXButton dashLogoutButton	Button that will log the user out
private JFXButton dashGetStartedButton	Button to send the user's cursor to the top search bar

private AnchorPane searchHistoryPane	The section for the search history
private AnchorPane atGlancePane	The section for the “At a Glance”
private AnchorPane favWordPane	The section for the Favorite word
private JFXButton addWordsShortcut	Button to send the user to the add word tab
private JFXButton removeWordsShortcut	Button to send the user to the customize word tab
private JFXButton openProfileShortcut	Button to send the user to the Help tab for Ducktionary tour (this was changed mid-development as the profile tab became the dashboard tab)
private JFXButton viewHelpShortcut	Button to send user to the help tab
private JFXButton wordOfTheDayShortcut;	Button to show the word of the day window

DatabaseFunction.java:

Important Functions:

Function Name	**Description**
public static void testConnect()	This method is checking whether connection between client and server database can be made. If the connection can’t be made, it will return the warning that the connection can’t be made.
public void addWordToDatabase (String a, String b, String c)	This method is for adding words to database. String a is for the word, String b is for the definition and String c is for the origin of the word. This method will make connection between client and server database and execute statement to add word in database.
public void getDataFromDatabase()	This method will get data from database. This method will make connection between client and server database and execute statement to load word from database.

public void deleteDataFromDatabase()	This method will delete data in database. This method will make connection between client and server database and execute statement to delete word from database.

Important Variables:

DefinitionPageController.java (JavaFX layout: DefinitionPage.fxml):

This function is responsible for displaying the first 3 definitions of a word corresponding to the selected part of speech.

Important Functions:

Function Name	Description
public void init(DuckHomeController duckHomeController)	Sets the DuckHomeController for this class, in order to access the home's variables and functions from the Definition Page.
public void resetColor()	Resets the text color for the places of use hyperlinks on the page. Used whenever a new word is loaded on the page, to display from which part of speech the definitions currently displayed are from.
void onPos1Clicked(ActionEvent event), void onPos2Clicked(ActionEvent event)	FXML event handlers for the part of speech hyperlinks in the page. When you click one of them, the clicked hyperlink turns green, the other one reverts to orange, and the definition corresponding to the part of speech is displayed on the definition labels.
void defFavoriteButtonGo(ActionEvent event)	This method is used to add the current word to the favorite word list if the user is logged on

Dictionary.java:

This class is responsible for reading the dictionary, storing it, and accessing its contents. Another custom class, EnglishWordLibrary, is used as part of this class.

Important Variables:

Variable	Description

private EnglishWordLibrary english	This handles parts of the file-reading process (such as the file location), as well as storing the raw data of the dictionary file, saved as a list of strings that contain each line.
private static String[] shortForms	This is a list of places of use (POS) forms as they are in the raw file, being lowercase and shortened. It's used to recognize the POS in the raw file to convert it to the longer forms.
private static String[] longForms	This is the extended form of the POS, which is the version that's meant to be displayed in the UI. The values in longForms that correspond to the shortened versions in shortForms are represented by the same index value. For example, "n" (short form) and "NOUN" (long form) are at the same index for their respective arrays.

Important Functions:

Function Name	**Description**
public ArrayList<ArrayList<String>> splitLine(String s)	This function takes a raw line from the EnglishWordLibrary, and splits it up into a 2D list of Strings to be returned. Index [0, 0] contains the word itself, [2, 0] contains the word's origin, and [1, n] contains the definitions, with each string containing a part of speech and all the definitions corresponding to that part of speech. So [1, 0] will contain the first POS and all the definitions in it, and if available, [1, 1] will contain the second POS with its definitions, and so on. Each definition String needs to be further split up with splitDefinition().
public ArrayList<String> splitDefinition(String s)	This function takes one of the definition String classes made by splitLine, and separates its contents into a list. The first value in the list is the part of speech for that set of definitions, and every other value is a definition for that word according to the POS.
public ObservableList<String>	This takes an EnglishWordLibrary and takes

returnWordList(EnglishWordLibrary wordLibrary), public ObservableList<String> returnWordList()	all the words (with no other information about the words) from the raw data, to be returned as an observable list. It's specifically used to display the words by themselves all at once. The parameter-less version uses the EnglishWordLibrary within the Dictionary class itself.
public ArrayList<ArrayList<String>> returnDisplayDictionary(String s), public ArrayList<ArrayList<String>> returnDisplayDictionary()	This takes a search parameter, and returns every word in the EnglishWordLibrary that contains the searched parameter. The returned words themselves are each a list containing the word itself, its places of use, its origin, and the first definition for each part of speech. The empty version will do the search without any parameters, which means that every word will be returned (the same thing happens if the user searches with an empty search bar).
public String returnWordLine(String toSearch, Dictionary dictionary), public String returnWordLine(String toSearch)	This takes a search parameter as a String (the word being searched), and a Dictionary class, to find if the searched word is within the Dictionary's EnglishWordLibrary file. If it is, the raw data for that particular word is returned, otherwise a dummy line ("nan") is returned. The version without a Dictionary parameter uses the Dictionary class calling the function.
public static ArrayList<String> readTextFile(File textFile)	This function reads a raw text file and returns its contents as a list into the EnglishWordLibrary. It's used in the EnglishWordLibrary constructor as opposed to the Dictionary itself. Some of the lines in the file are empty, so those lines aren't read. In particular, there's a line with garbage data that can't be removed, so the line in the raw data is always removed.

DuckHomeController.java (JavaFX layout: DuckHome.fxml):

This class handles the program's entire sidebar and top panels (where the search bar and logo is located). The rest of the program's menus are always contained within this class, and as such, everything in this class is always accessible regardless of where the user is in the program.

Important Functions:

Function Name	Function Description
void handleKeyOnReleased():	This function is responsible for all keyboard inputs in the program. It is currently used to bind keyboard shortcuts (F1 through F4) to transport the user between the dictionary's main menus.
public void goToDefinitionPage(String wordKey)	This function is called whenever the definition page needs to be accessed for a certain word. The function accesses a word with wordKey, gets all of its information and displays it using the Labels and Hyperlinks in the DefinitionPage. Updating all the labels in DefinitionPage is done using separate functions in DuckHomeController that are used in this function. Afterwards, it switches into the DefinitionPage to display the word's information.
public void setSearchBarOption()	This updates the list of words as the search bar's autocomplete depending on what's in the search bar, and displaying it.
public void search()	This handles what happens if the user searches with the search bar (by pressing the icon beside it). If the word they searched is in the dictionary, they will go to the definition page for that word. Otherwise, they will go to a notification page that tells them the word doesn't exist, and recommends similar words depending on what they searched.
public void setViewSearchedWordsPageSelectedAlphabets(String alphabets)	This updates the list of words in the ViewSearchedWordsPage depending on what range of the alphabet is currently selected.

Important Variables:

Standard Variables	Description

private Dictionary dictionary	This variable stores the Dictionary class used for the program, which stores all the words, places of use, definitions, and origins. It also has functions to access certain words in the dictionary and returning it in readable form.
private ObservableList<String> wordList	This observable list stores a displayable form of a list of every word in the dictionary, to be used as a cleaner way to retrieve the displayable dictionary.
private ArrayList<ArrayList<String>> definitionContent	This is used to store a word's information (its definition and places of use) while it is currently being displayed in the DefinitionPage. It needs to be part of the class because the program needs to change the definitions of the word depending on what part of speech is currently selected for that word. To do that, a copy of that information needs to be stored in DuckHome so it can be quickly accessed and used to replace the displayed definitions on the DefinitionPage class.
private ObservableList<String> history	This stores what words are currently displayed as the user's search history.
private ObservableList<String> favorite	This stores what words are currently displayed as the words the user has favorited.
private ObservableList<String> atAGlance	This stores what words are currently displayed as a list of random words based on the date set on the user's computer.

EnglishWordLibrary.java:

This class is contained in the Dictionary class, being used to store the raw data of the dictionary text file to be read by a Dictionary object.

Important Functions:

Function Name	**Description**

public void readFile(), public void readFile(String newLocation)	This reads the dictionary text file placed in the newLocation file location, and stores every line into EnglishWordLibrary's list using Dictionary's readTextFile() function. The EnglishWordLibrary's fileLocation variable will be used as the file location instead if using the parameterless version of the function.

Important Variables:

Variables	**Description**
private File file	This File is where the dictionary's txt file will be loaded into the program.
private String fileLocation	Contains the file location address of the txt file.
private ArrayList<String> list	Contains the list of every line contained in the text file, with each line representing a word and all of its contents in the dictionary.

Funfact.java:

This class shows the fun fact of the day according to the PC's date settings. The fun fact is shown on the splash screen when the application splash is showing.

Important Functions:

Function Name	**Description**
ArrayList<String> getFunfact()	Gets the fun fact arraylist
setFunfact(ArrayList<String> funfact)	Set the current fun fact from the array list
Funfact()	Adds all the fun facts into the funfact arraylist
String getFunFact()	Gets the current date from the calendar, sets it into the date variable and then returns the fun fact for the day from the arraylist according to the date from the calendar.

Important Variables:

Variables	**Description**

ArrayList<String> funfact = new ArrayList<>()	A list of fun facts that can be displayed. The array's contents are set in the class constructor Funfact().
int date = cal.get(Calendar.DATE)	Gets the current date to be used to select the funfact (Date 1 = index 0)

HelpPageController.java (JavaFX layout: HelpPage.fxml):

This page is responsible for controlling the HelpPage.fxml file which acts as the help tab in our application. This controller is basically responsible for opening up new stages (windows) that contain the appropriate fxml for each help screen.

Important Functions:

Function Name	Description
void launchHelpWindow(String helpFXMLFile)	Provides the functionality of launching a new window that has no top bar. Other functions that opens a window in the help tab will use this function and fill the string with their respected fxml files. This function also handles all the garbage collection necessary to prevent memory leak.
void helpDucktionaryTour(ActionEvent event)	Opens the Ducktionary tour window
void helpHowToSearch(ActionEvent event)	Opens the How to Search Help window
void helpAddWord(ActionEvent event)	Opens the Add Word Help window
void helpRemoveWord(ActionEvent event)	Opens the Remove Word Help window
void helpTipsAndTricks(ActionEvent event)	Opens the Tips and Tricks Help window
void helpKeyboardShortcut(ActionEvent event)	Opens the Keyboard Shortcuts Help window
void helpFAQ(ActionEvent event)	Opens the FAQ window
void helpSignIn(ActionEvent event)	Opens the Sign In Help window
void helpUserManual(ActionEvent event)	Opens the User Manual folder location in the application directory
void helpProgramManual(ActionEvent event)	Opens the Program Manual folder location in the application directory

private void openDocumentationFolderInExplorer(String folderName)	This function is the base function that opens the location of the directory in explorer. All the methods that opens directories will use this function and fill up the directory in the string parameter.
void helpManageSettings(ActionEvent event)	Opens the Settings Help window
void helpAbout(ActionEvent event)	Opens the About window

Important Variables:

Variables	**Description**
private JFXButton help1Button;	button for Ducktionary Tour
private JFXButton help2Button;	button for How to Search
private JFXButton help3Button;	button for How To Add a New Word
private JFXButton help4Button;	button for How to remove a word
private JFXButton help5Button;	button for Tips and Tricks
private JFXButton help6Button;	button for Keyboard Shortcuts
private JFXButton help7Button;	button for FAQ
private JFXButton help8Button;	button for Sign in Help
private JFXButton help9Button;	button for View user Manual
private JFXButton help10Button;	button for View Program Manual
private JFXButton help11Button;	button for Manage Settings
private JFXButton help12Button;	button for About Page
public label labelBig	Label for the top description

HelpAboutController (JavaFX layout: HelpAbout.fxml):

This class is responsible for controlling the “About” window that is displayed when clicking the About section in the help menu.

Important Functions:

Function Name	**Description**

public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. Run when the close button is clicked.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
private Pane pane	Pane representing the entire About window
private JFXButton closeButton	Button used to close the window

HelpAddWordController (JavaFX layout: HelpAddWord.fxml):

This is responsible for controlling the window that pops up when the "How to Add a New Word" tutorial is clicked in the help menu.

Important Functions:

Function Name	**Description**
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable. It's also used to set the add word MediaPlayer into the media1

	MediaView object, and making it play automatically.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. The mediaPlayerAddWord object is disposed of before the garbage collection so it will be removed and cleared from the RAM. Runs when the close button is clicked.
public void playVideo1(MouseEvent event)	Plays the video file in the MediaView if it is clicked.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
String pathVideoAddWord	Contains the file path to the "add new words" tutorial video
Media mediaAddWord	Contains the Media taken from pathVideoAddWord file path. The file itself is
MediaPlayer mediaPlayerAddWord	Contains the MediaPlayer object containing the Media file, to be displayed in MediaView inside the FXML.
private AnchorPane anchorPane	Pane representing the entire add word window

private JFXButton closeButton	Button used to close the window
private MediaView media1	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerAddWord.

HelpDucktionaryTourController(JavaFX layout: HelpDucktionaryTour.fxml):

Responsible for the window opened when clicking the ducktionary tour option, giving a general rundown on how to use the dictionary.

Important Functions:

Function Name	Description
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable. It's also used to set the MediaPlayer objects into the MediaView objects.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. The mediaPlayerAddWord objects are disposed of before the garbage collection so they will be removed and cleared from the RAM. Runs when the close button is clicked.
public void playVideoCustomize(MouseEvent event)	Plays the customize menu tour video in the mediaCustomize view when it's clicked on.
public void playVideoHelp(MouseEvent event)	Plays the help menu tour video in the mediaHelp view when it's clicked on.
public void playVideoProfile(MouseEvent event)	Plays the profile (in home menu) tour video in the mediaProfile view when it's clicked on.

public void playVideoSearch(MouseEvent event)	Plays the search menu tour video in the mediaSearch view when it's clicked on.
public void playVideoHome(MouseEvent event)	Plays the home menu tour video in the mediaHome view when it's clicked on.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
String pathVideoSearch	Contains the file path to the "search" tutorial video
Media mediaSearchVideo	Contains the Media object taken from pathVideoSearch file path. The file itself is an mp4.
String pathVideoHome	Contains the file path to the "home" tutorial video
Media mediaHomeVideo	Contains the Media object taken from pathVideoHome file path. The file itself is an mp4.
String pathVideoHelp	Contains the file path to the "help" tutorial video
Media mediaHelpVideo	Contains the Media object taken from pathVideoHelp file path. The file itself is an mp4.
String pathVideoCustomize	Contains the file path to the "customize" tutorial video
Media mediaCustomizeVideo	Contains the Media object taken from pathVideoCustomize file path. The file itself is an mp4.
String pathVideoProfile	Contains the file path to the "profile" tutorial video

Media mediaProfileVideo	Contains the Media object taken from pathVideoProfile file path. The file itself is an mp4.
MediaPlayer mediaPlayerHome	Contains the MediaPlayer object containing the home Media file, to be displayed in a MediaView inside the FXML.
MediaPlayer mediaPlayerHelp	Contains the MediaPlayer object containing the help Media file, to be displayed in a MediaView inside the FXML.
MediaPlayer mediaPlayerSearch	Contains the MediaPlayer object containing the search Media file, to be displayed in a MediaView inside the FXML.
MediaPlayer mediaPlayerProfile	Contains the MediaPlayer object containing the profile Media file, to be displayed in a MediaView inside the FXML.
MediaPlayer mediaPlayerCustomize	Contains the MediaPlayer object containing the customize Media file, to be displayed in a MediaView inside the FXML.
private AnchorPane anchorPane	Pane representing the entire Tour window
private JFXButton closeButton	Button used to close the window
private MediaView mediaHome	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerHome.
private MediaView mediaSearch	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerSearch.
private MediaView mediaCustomize	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerCustomize.
private MediaView mediaProfile	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerProfile.
private MediaView mediaHelp	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerHelp.

private Accordion accordion	Container for all of the titled panes in the window, clicking on a titled pane displays the content inside it below.
private TitledPane titledPaneHome	Titled pane for the home tutorial section of this tour, includes mediaHome inside
private TitledPane titledPaneSearch	Titled pane for the search tutorial tab, includes mediaSearch inside
private TitledPane titledPaneCustomize	Titled pane for the customize tutorial section of this tour, includes mediaCustomize inside
private TitledPane titledPaneProfile	Titled pane for the profile tutorial section of this tour, includes mediaProfile inside
private TitledPane titledPaneHelp	Titled pane for the help tutorial section of this tour, includes mediaHelp inside

HelpFAQController (JavaFX layout: HelpFAQ.fxml):

Responsible for the window that pops up when the FAQ option is selected, which answers a set of general questions that users may ask while using the dictionary.

Important Functions:

Function Name	Description
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. Runs when the close button is clicked.

Important Variables:

Variables	Description
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
private AnchorPane anchorPane	Pane where the entire FAQ window is contained
private JFXButton closeButton	Button used to close the window

HelpKeyboardShortcutController (JavaFX layout: HelpKeyboardShortcut.fxml):

Important Functions:

Responsible for the window that shows up when picking the keyboard shortcut option in the help menu, for a list of keyboard shortcuts.

Function Name	Description
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. Runs when the close button is clicked.

Important Variables:

Variables	Description

private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
private AnchorPane anchorPane	Pane where the entire keyboard shortcut window is contained
private JFXButton closeButton	Button used to close the window

HelpRemoveWordController (JavaFX layout: HelpRemoveWord.fxml):

Important Functions:

Handles the window that pops up when the "How to remove a word" button is clicked on the help page, telling users how to remove a word from the dictionary.

Function Name	**Description**
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable. It's also used to set the add word MediaPlayer into the media1 MediaView object, and making it play automatically.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. The mediaPlayerAddWord object is disposed of before the garbage collection so it will be removed and cleared from the RAM. Runs when the close button is clicked.

public void playVideo1(MouseEvent event)	Plays the video file in the MediaView if it is clicked.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
String pathVideoAddWord	Contains the file path to the "remove words" tutorial video
Media mediaAddWord	Contains the Media taken from pathVideoAddWord file path. The file itself is
MediaPlayer mediaPlayerAddWord	Contains the MediaPlayer object containing the Media file, to be displayed in MediaView inside the FXML.
private AnchorPane anchorPane	Pane representing the entire add word window
private JFXButton closeButton	Button used to close the window
private MediaView media1	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerAddWord.

HelpSearchController (JavaFX layout: HelpSearch.fxml):

Important Functions:

Handles the window that pops up when the "How to search" button is clicked, telling users how to search for words.

Function Name	**Description**
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making

	the window draggable. It's also used to set the MediaPlayer objects into the MediaView objects.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. The MediaPlayer objects are disposed of before the garbage collection so they will be removed and cleared from the RAM. Runs when the close button is clicked.
public void playVideoHowSearch(MouseEvent event)	Plays the how-to search video in the mediaPlayerHowSearch view when it's clicked on.
public void playVideoViewWord(MouseEvent event)	Plays the view word video in the mediaPlayerViewWord view when it's clicked on.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
private AnchorPane anchorPane	Pane representing the entire search help window
private JFXButton closeButton	Button used to close the window

String pathVideoSearchWords	Contains the file path to the “search words” tutorial video
Media mediaSearchWords	Contains the Media object taken from pathVideoSearchWords file path. The file itself is an mp4.
String pathVideoViewWord	Contains the file path to the “view word” tutorial video
Media mediaViewWordVideo	Contains the Media object taken from pathVideoViewWord file path. The file itself is an mp4.
MediaPlayer mediaPlayerHowSearch	Contains the MediaPlayer object containing the mediaSearchWords file, to be displayed in MediaView inside the FXML.
MediaPlayer mediaPlayerViewWord	Contains the MediaPlayer object containing the mediaViewWordVideo file, to be displayed in MediaView inside the FXML.
private MediaView mediaHowSearch	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerHowSearch.
private MediaView mediaViewWord	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerViewWord.

HelpSignInController (JavaFX layout: HelpSignIn.fxml):

Important Functions:

Handles the window that pops up when the “sign-in help” button is pressed, telling the user how to sign in, register, and log out.

Function Name	**Description**
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It’s used to run setDraggableWindow(), making the window draggable. It’s also used to set the MediaPlayer objects into the MediaView objects.

public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. The MediaPlayer objects are disposed of before the garbage collection so they will be removed and cleared from the RAM. Runs when the close button is clicked.
public void playVideoSignIn(MouseEvent event)	Plays the how-to sign in video in the mediaPlayerSignIn view when it's clicked on.
public void playVideRegister(MouseEvent event)	Plays the view word video in the mediaPlayerRegister view when it's clicked on.
public void playVideoLogout(MouseEvent event)	Plays the view word video in the mediaPlayerLogout view when it's clicked on.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
String pathVideoLogin	Contains the file path to the "login" tutorial video
Media mediaLogin	Contains the Media object taken from pathVideoLogin file path. The file itself is an mp4.

String pathVideoRegister	Contains the file path to the “register” tutorial video
Media mediaRegisterVideo	Contains the Media object taken from pathVideoRegister file path. The file itself is an mp4.
String pathVideoLogout	Contains the file path to the “logout” tutorial video
Media mediaLogoutVideo	Contains the Media object taken from pathVideoLogout file path. The file itself is an mp4.
MediaPlayer mediaPlayerSignIn	Contains the MediaPlayer object containing the mediaLogin file, to be displayed in MediaView inside the FXML.
MediaPlayer mediaPlayerRegister	Contains the MediaPlayer object containing the mediaRegisterVideo file, to be displayed in MediaView inside the FXML.
MediaPlayer mediaPlayerLogout	Contains the MediaPlayer object containing the mediaLogoutVideo file, to be displayed in MediaView inside the FXML.
private MediaView mediaHowSearch	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerSignIn.
private MediaView mediaViewWord	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerRegister.
private MediaView mediaLogout	Container for a MediaPlayer object to be displayed in the FXML, uses mediaPlayerLogout.
private AnchorPane anchorPane	Pane representing the entire add word window
private JFXButton closeButton	Button used to close the window

HelpTipsAndTricksController (JavaFX layout: HelpTipsAndTricks.fxml):

Important Functions:

Handles the window that pops up when the tips-and-tricks button is pressed, giving the user some hints on how to use the program.

Function Name	**Description**
public void initialize(URL location, ResourceBundle resources)	Runs when the window is first opened. It's used to run setDraggableWindow(), making the window draggable.
public void setDraggableWindow()	This function makes the window draggable by clicking on it from anywhere, by adding OnMousePressed() and OnMouseDragged() events for the current pane. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.
public void closeDuckTour(ActionEvent event)	This closes the window and clears the ram using garbage collection. Runs when the close button is clicked.

Important Variables:

Variables	**Description**
private double xOffset	The window's current x location when clicked. Used to offset the window movement when dragged in horizontal directions.
private double yOffset	The window's current y location when clicked. Used to offset the window movement when dragged in vertical directions.
private AnchorPane anchorPane	Pane representing the entire add word window
private JFXButton closeButton	Button used to close the window

LoginPageController.java (JavaFX layout: LoginPage.fxml):

This page handles all of the login

Important Functions:

Function Name	Description
void makeLogin(ActionEvent event)	Gets all the user input, validates them if they are correctly inputted and then signs the user into the database before then sending the user back to the login page.
void makeSignup(ActionEvent event)	Switches the current scene to the signup scene
public void init(DuckHomeController duckHomeController)	Make duckHomeController as mother

Important Variables:

FXML Variable	Description
public JFXTextField user;	The field used to input username
public JFXPasswordField password;	The field used to input user password
JFXButton signup;	The button used to signup by executing the makeSignup() method
JFXButton login;	The button used to login by executing the makeLogin() method
ImageView imageBeard;	Image of duck logo
AnchorPane loginPane;	The current scene of login page

Main.java:

This is the main class of the dictionary program. One of the best practices guidelines of Java for JavaFX programs is to use the main class only to start up the standalone application. This class is a subclass of the javafx.application.Application class.

Functions:

Function Name	Description
public void start(Stage primaryStage)	This is an overridden function from the javafx.application.Application class, running once the JavaFX application starts. It is used to start up the JavaFX program, by calling the DuckHome.fxml, loading it into a Scene, and displaying it. An error message will be shown

	instead if the DuckHome.fxml is missing from the program.
public static void main(String[] args)	The main function that runs when the program starts. It is only used to run a standalone application with launch(), specifically our JavaFX application.

OnboardingLauncherController.java & OnboardingController.java:

The OnboardingLauncherController.java is responsible for the OnboardingSlide1.fxml file while the OnboardingController.java is responsible for the OnboardingSlide1.fxml, OnboardingSlide2.fxml, OnboardingSlide3.fxml, OnboardingSlide4.fxml, and OnboardingSlide5.fxml file.
These controllers are responsible for displaying and animating all of the slides for the onboarding (the tutorial on start-up). In particular, the first slide is handled by OnboardingLauncherController, while the other 4 slides are handled by OnboardingController. To launch the onboarding, this class is called, but for the rest of the onboarding slides, OnboardingController is used instead. It crashes if you click on bubble 1 but all of them use a single controller OnboardingLauncherController is the frame for where all the fxmls get inserted, the fxmls are actually loaded and animated with OnboardingController.

Important Functions:

Function Name	**Description**
public void initialize(URL location, ResourceBundle resources)	A function that runs when the slide is first loaded. It's used to call setDraggableWindow().
private void animateSwipe(Parent root, Scene scene)	This function is used to change the current slide on the onboarding and animate the movement its as it transitions between slides. Changing the slide is done by taking the root, which is the slide that is being transitioned to, and the scene, which is the slide being transitioned from, and animating by adding the root to the scene, and sliding the scene from the original scene to the root scene. The original scene is unloaded after the animation finishes.
void goToSlide1(MouseEvent event), void goToSlide2(MouseEvent event), void goToSlide3(MouseEvent event), void	These are all event handlers for each of the clickable Circle classes on the onboarding, which are used to represent each slide in the

goToSlide4(MouseEvent event), void goToSlide5(MouseEvent event)	onboarding. Each function corresponds to each of the 5 slides, which loads their particular slide and calls animateSwipe() to transition and change into the slide.
public void setDraggableWindow()	This adds OnMousePressed() and OnMouseDragged() events for the current slide pane. It allows the onboarding to be moved around by dragging it regardless of where the user started clicking it on. So when the background of the onboarding is clicked, it will get the current location of the pointer relative to the onboarding, and when dragged, it will move the onboarding around the screen, offset by the pointer location.

Important Variables:

Variable Name	Description
private double xOffset, private double yOffset	These variables are used to determine how far away the onboarding is from its initial position. It's used to reposition the onboarding without having to use a window frame and being able to drag it by clicking anywhere on the onboarding (see setDraggableWindow() for more details).
private StackPane parentContainer	Only appearing in OnboardingLauncherController (slide 1), this acts as the bottom-most layer of the FXML, containing every other FXML component.

Profile.java:

Important Functions:

Important Variables:

RegisterPageController.java (JavaFX layout: RegisterPage.fxml):

This controller is responsible for handling the register page including getting the forms, validating the forms, and then putting the information into the database.

Important Functions:

Function Name	Description
void makeLogin(ActionEvent event) {	Changes the current scene into the login tab
void makeSignup(ActionEvent event) {	Gets all the user input, validates them if they are correctly inputted (if the password is there, username is there, proper naming, etc) and then signs the user up into the database before then sending the user back to the login page
public void init(DuckHomeController duckHomeController) {	Initalizes mother as duckHomeController.

Important Variables:

FXML Variable	Description
public JFXTextField user;	Textfield for the username
public JFXTextField email;	Textfield for the user email
public JFXPasswordField password;	Textfield for the user password
public JFXPasswordField passwordConfirm;	Textfield for the user password
JFXButton signup;	The button used to signup and execute the makeSignUp() method
JFXButton login;	The button used to cancel the registration process and switch the user back to the login screen using the makeLogin() method.
AnchorPane registerPane;	The current anchorpane scene

SearchPageController.java (JavaFX: SearchPage.fxml):

This class handles the search page, which allows the user to see the list of words in the dictionary, separated by what letters they start with in alphabetical order. Specifically, this displays the list of alphabet ranges (represented by buttons) that the user can select before viewing the list.

Important Functions:

Function Name	Description
void Clicked()	Functions such as abcClicked(), and

	defClicked all serve the same purpose, namely setting the range of letters in the dictionary before going into the list of words.

Important Variables:

FXML Variable	Description
private DuckHomeController mother	This connects the SearchPageController class with the main panel (the DuckHomeController class), allowing access to its functions and variables.

SplashController.java:

Responsible for the splash screen that shows up at the start of the program.

Important Functions:

Function Name	Description
public void initialize(URL location, ResourceBundle resources)	Runs when the splash screen is first opened. It fades in the splash screen, sets up the animation timeline, and prepares to fade out after the animation timeline finishes.
private void update()	This function handles updating the percentage number and ending the animation timeline once it reaches 100%.

Important Variables:

Variables	Description
private Timeline timeline	Animation timeline for the loading percentage number.
private FadeTransition fadeIn	Fade transition that transitions into appearing the window.
private FadeTransition fadeOut	Fade transition that transitions into making the window disappear.
private Label funfactlabel	Where a random fun fact from the Funfact object is displayed.
private AnchorPane rootAnchor	Anchor pane containing the entire window.

private JFXSpinner progressPercentageSpinner	A spinner that animates for the loading percentage number.
private Label progressPercentageLabel	The displayed percentage number

ViewSearchedWordsPageController.java (JavaFX: ViewSearchedWordsPage.fxml):

This class handles displaying the list of words from the dictionary according to what range of letters were chosen from the SearchPageController.

Important Functions:

Function Name	Description
void init()	This function is ran when the program is first started in DuckHomeController. Unlike some other classes, which only use this function to connect the class with the DuckHomeController, this also sets up the value of the columns in the TableView, and setting a click listener for the rows in tableview that sends the program into the DefinitionPage upon double-clicking a word, with the information of the word being loaded into the DefinitionPage.
void updateList(String range)	This class updates the list of words that are displayed as a table on the page. In particular, it checks the range of alphabets that are selected, and only includes words that are within that range (a range of "ABC" will only include words that start with A to C). The function is called in SearchPageController, right before going into the ViewSearchedWordsPage.

Important Variables:

FXML Variable	Description
private DuckHomeController mother	This connects the ViewSearchedWordsPageController class with the main panel (the DuckHomeController class), allowing access

	to its functions and variables.
public TableColumn<String, String> columnWord	The value of the column's displayed word is taken directly from the items in the TableView (which contains String values).

WordNotFoundPageController.java (JavaFX layout: WordNotFoundPage.fxml):

This class handles when the user searches a word that doesnt exist in database. When a user searches a word that doesnt exit the user is offered to add that word.

Important Functions:

Function name :	Description
void addWordGo	Tells mother controller to switch tab to add word page
public void init	Sets the mother controller as duck home controller

Important Variables:

FXML Variable	Description
private DuckHomeController mother	This is the mother controller
private JFXButton buttonAddWord;	Button to go to add word page

WordOfTheDayPageController.java (JavaFX layout: WordOfTheDayPage.fxml):

This controller is responsible for the window that displays the word of the day, accessed by clicking the text on the bottom of the dashboard page.

Important Functions:

Function Name	Description
void addWordGo	Tells mother controller to switch tab to add word page
public void initialize(URL arg0, ResourceBundle arg1)	Function that runs when the window is opened. It gets the current date and randomly selects a word from the wordList, and set all of the relevant labels to display the information (display the date on the "date" Label, "wordName" label is set the word

	itself, and so on)
public void init	Sets the mother controller as duck home controller
public ArrayList<ArrayList<String>> splitLine(String s)	This function takes a raw line from the wordList, and splits it up into a 2D list of Strings to be returned. Index [0, 0] contains the word itself, [2, 0] contains the word's origin, and [1, n] contains the definitions, with each string containing a part of speech and all the definitions corresponding to that part of speech. So [1, 0] will contain the first POS and all the definitions in it, and if available, [1, 1] will contain the second POS with its definitions, and so on. Each definition String needs to be further split up with splitDefinition(). Identical to Dictionary.java's splitLine().
public ArrayList<String> splitDefinition(String s)	This function takes one of the definition String classes made by splitLine, and separates its contents into a list. The first value in the list is the part of speech for that set of definitions, and every other value is a definition for that word according to the POS. Identical to Dictionary.java's splitDefinition().

Important Variables:

Variables	**Description**
private DuckHomeController mother	This connects this class with the main panel (the DuckHomeController class), allowing access to its functions and variables.
String wordList[]	A list of words and their information (parts of speech, origin, definitions), formatted identically as the raw data contained in an EnglishWordLibrary.list variable. A word is randomly selected to display every time the window is opened.
private Label date	The current date displayed
private Label wordName	Display of the current word itself

private Label wordType	Display of the word type (first parts of speech)
private Label wordDefinition	Display of the word's first definition

Part 4: More Information

More information about Ducktionary such as the user manual or class diagram can be found on the Ducktionary install folder.

Support

The support team and the development teams consist of the following people:

James Adhitthana (00000021759) [Coordinator]

Andre Kurnia (00000021269)

Andreas Geraldo (00000021533)

Deananda Irwansyah (00000025513)

Regy Ezananta (00000026169)

Thompson Dharmawan (00000022386)