|  |  |  |  |
| --- | --- | --- | --- |
| **CAPSTONE PROJECT REPORT** | | | |
| **GFM: An enhanced Grammatical Framework minibar** | | | |
| **22-29 September 2023** | | | |
| Client | **Zola Mahlaza** [**zola.mahlaza@uct.ac.za**](mailto:zola.mahlaza@uct.ac.za) | Tutor | **Hamza Amir** [**amrham001@myuct.ac.za**](mailto:amrham001@myuct.ac.za) |
| Team Members | **KPPDEA001, Dean Kopping,** [**kppdea001@myuct.ac.za**](mailto:kppdea001@myuct.ac.za) | | |
|  | **FRNMOD001, Modjadji Francis,** [**frnmod001@myuct.ac.za**](mailto:frnmod001@myuct.ac.za) | | |
|  | **TSHCAS005, Cassandra Tshabalala,** [**tshcas005@myuct.ac.za**](mailto:tshcas005@myuct.ac.za) | | |

**Table of Contents**

[**Introduction** 3](#_Toc146573195)

[About 3](#_Toc146573196)

[Problem 3](#_Toc146573197)

[Scope Statement 3](#_Toc146573198)

[Proposed Solution 4](#_Toc146573199)

[**Requirements Analysis** 5](#_Toc146573200)

[Functional requirements 5](#_Toc146573201)

[**Nonfunctional requirements**  11](#_Toc146573202)

[**Usability Requirements**  12](#_Toc146573203)

[**Design Overview** 13](#_Toc146573204)

[**Discussion of Analysis Artefacts Produced** 13](#_Toc146573205)

[**Architecture Pattern** 13](#_Toc146573206)

[**Architecture Overview** 13](#_Toc146573207)

[Algorithms and Data Organization 14](#_Toc146573208)

[Design Class Diagram 14](#_Toc146573209)

[**Implementation** 15](#_Toc146573210)

[**Program Validation and Verification** 17](#_Toc146573211)

[**Quality Management Plan** 17](#_Toc146573212)

[Product needs to agree with the client requirements, Validation Testing 17](#_Toc146573213)

[Testing Protocol (make page Landscape) 18](#_Toc146573214)

[Product needs to make use of well-defined classes and methods, Class Testing 21](#_Toc146573215)

[**Discussion of Results** 22](#_Toc146573216)

[**Conclusion** 22](#_Toc146573217)

[**User Manual** 23](#_Toc146573218)

[**References** 24](#_Toc146573219)

# **Introduction**

About

Grammatical Framework is a programming language used for multilingual grammar applications [reference]. The framework has many uses, some of which include building translation systems, multilingual web gadgets, natural-language interfaces, and dialogue systems. The cloud-based editor: Grammatical Framework Minibar is one of the implementations of the grammatical framework. It was an implementation of Fridge Poetry, a GF web application, developed by Hallgreen, Enache and Ranta.

Problem

Minibar Online’s currentprimary functions are to author and translate user/random text as per a specific controlled natural language using a chosen grammar file. The translation of the input text is then displayed. Some of the issues associated with the current functionality are displayed below:

1. The application does not provide offline functionality.
2. A user is restricted to the pre-loaded grammar files on the application and cannot use their own grammar file.
3. A user cannot translate more than one sentence at a time. The current Minibar allows the user to translate a single sentence at a time. Once a user inputs a period, they are unable to translate another sentence.
4. The application does not make use of easy/intuitive deletion of words

Scope Statement

The project's main purpose is to develop an enhanced version of this cloud-based editor. To do so, the following features are to be implemented:

1. **Offline functionality:** The enhanced version is to run locally, operate without any internet access and should not be part of an integrated development environment.
2. **The ability to upload their own grammar files:** Users of the enhanced system need to be able to upload their own grammar files to use when authoring text.
3. **Translate multiple sentences:**  User can translate multiple sentences at a time (i.e., the ability to translate paragraphs)
4. **Easy deletion of words:** Users need a simpler way of deleting words in the textbox instead of relying on the provided deletion buttons. Additionally, with the third feature being incorporated, this requirement expands to allowing easy deletion for all sentences in the case of translating multiple sentences.
5. **Predictive typing:** The user’s historical information is to be used for predictive typing. This means that as a user input their phrase in the minibar, the suggested next words need to be ordered based on the user’s history with the most recently used words being ordered first.

In addition to the above requirements, the project team aims to improve the usability and user experience of the application, while remaining in line with the given requirements by applying principles of design to improve aspects of the application using tools such as Norman’s 7 Principles of Design. This will allow for an improved and more user-friendly interface. These additional improvements are the secondary features of the application. Continuously, the project’s core goal is to fulfil all the client’s requirements efficiently and fully by achieving a realistic balance between the predetermined scope, budget, and time of the project.

Proposed Solution

To enhance the current application functionality, the project team decided to implement a desktop version of the application incorporating all the additional features (as required by the client and as defined in the project scope). This was achieved in the following manner:

1. **Offline functionality:** 
   1. By making the application a desktop application, MiniBar will now be fully functional offline as it does not require any online access to render user translations. A desktop application was developed using JavaFX and Scene Builder.
   2. To achieve this functionality, a Waterfall software development method was adopted. The user requirements for this functionality were gathered through client meetings where the criteria to judge the functionality against was outlined. This criterion consisted of the following:
      1. A user can interact with the system without any internet access.
      2. User translations could occur offline
   3. After requirements gathering, the system’s offline functionality was designed and implemented using the IntelliJ IDE and all relevant modules and libraries.
2. **Uploading Own Grammar Files**
   1. Unlike the original functionality of the application, users are now able to upload their own grammar files for rendering translations. This is achieved via an upload button which allows a user to access their local device storage and choose their desired .PGF file. This is then loaded onto the application and displayed accordingly.
3. **Translating Multiple Sentences**
   1. To allow users the ability to translate multiple sentences, the application implements more than one input box. This allows the user to enter multiple phrases to translate at a time, with each input box being tied to a panel displaying the translation of the entered phrase in the chosen translation language.
   2. **SE Method**
4. **Easy Deletion of Words** 
   1. To provide an intuitive deletion of words, users are able to make use of the keyboard’s delete/backspace functionality. This approach is intuitive as it is something that users are familiar with.
5. **Predictive Typing**
   1. To allow users to have better options in the texts that are predicted. The predicted words are based on what they have given as input in the past.

**Software Engineering Method**

To successfully implement the overall functionality of the application, a Waterfall Development method was majorly used, with some aspects of agility. The project scope was well determined in the beginning of the project and all the requirements were gathered and outlined initially, indicating a waterfall approach, while the project time and budget were fixed (elements of Agile Development). Once the client outlined their desired functionality, each was analysed and designed. Before implementation, the initial designs of the system were shared with the client for approval (an Agile element of customer involvement), and necessary changes were made. This process of gathering requirements, analysing and designing and then implementing was utilised for each of the features of the application.

# **Requirements Analysis**

## Functional requirements

In order to clearly define the functional requirements of the product, the project team created a use case diagram (*Appendix A)* accompanied by use case narratives corresponding to each product functionality to model the user’s expected interaction and how the system responses in each instance. Through modelling use cases, the requirements and expected behaviour were easily outlined and through contact with the client, were verified.

Below are the use case narratives related to each application feature which were used to validate the functional requirements:

**Uploading own Grammar File**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Uploading own PGF Grammar File | **ID:**1 | | **Importance:** High |
| **Primary Actor:** User | | | |
| **Brief Description:** The actor can upload their own .PGF file from their local device to use for authoring translations. | | | |
| **Preconditions:**   1. The user needs to have MiniBar Offline installed on their local device | | | |
| **Postconditions:**   1. A grammar file must be uploaded, and its contents displayed | | | |
| **Related Use Cases:**   1. None | | | |
| **Normal Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. The user opens the application on their device | | 1. App launch | |
| 1. The user navigates to the PGF grammar file section and clicks on the **Upload** button | | 1. A pop-up window to the user’s local device storage opens | |
| 1. Selects the .PGF file they would like to upload for authoring translations | | 1. The file is uploaded as the PGF file to be used for authoring and its contents are displayed | |
| **Alternate/Exceptional Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. No .PGF file available on the user’s local storage | | 1. Will not permit the upload of any other file format | |

**Changing Translation Settings**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Modify translation settings | **ID:** 2 | | **Importance:** High |
| **Primary Actor:** User | | | |
| **Brief Description:** The user can modify the app’s translation settings. This includes being able to choose an input and output language and selecting a translation category. | | | |
| **Preconditions:** None | | | |
| **Postconditions:**   1. The user has chosen an input and output language for their translation 2. The user has selected a category for translation. | | | |
| **Related Use Cases: None** | | | |
| **Normal Flow of Events** | | | |
| **Flow of Events** | | **System Response** | |
| 1. The user navigates to the input language section and selects their input language | | 1. Input language is updated | |
| 1. The user navigates to the output language section and selects their output language | | 1. Output language is updated | |
| 1. The user selects the category of input they will be providing the application | | 1. Input category is updated | |
| **Alternate/Exceptional Flow of Events** | | | |
| **Flow of Events** | | **System Response** | |
| 1. The user decides to not change language settings | | 1. The default input and output language and category are used to author the translation | |

**Rendering Multiple Translations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Input sentence(s) to be translation | **ID:** 3 | | **Importance:** High |
| **Primary Actor:** User | | | |
| **Brief Description:** The user can input text they would like to be translated. This can be in the form of a single sentence or multiple sentences. | | | |
| **Preconditions:** For multiple sentences, a user is able to input up to three sentences using the multiple textboxes provided for each sentence. | | | |
| **Postconditions:** User enters what they would like to have translated | | | |
| **Related Use Cases:**   1. **Include –** Verify Input 2. **Extend –** Select Random | | | |
| **Normal Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. User navigates to the text input areas and inputs text they would like to translate. | | 1. Application receives and processes inputted text [Include: verify input] and verifies whether the text is valid in terms of formation and restrictions or not. | |
| 1. User gives indication to translate inputted text | | 1. Application commences translation | |
| **Alternate/Exceptional Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. User selects random to generate random text | | 1. Generates random text to be used as input for the translation. | |
| 1. User enters a word not part of the PGF file | | 1. Error message | |

**Intuitive Deletion**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Delete text/words | **ID:** 4 | | **Importance:** High |
| **Primary Actor:** User | | | |
| **Brief Description:** The user can delete words or phrases from specific sentences that they have inputted. Delete is intuitive and linked to sentences. | | | |
| **Preconditions:** There must be words to delete | | | |
| **Postconditions:** Words that the user would like to be removed are easily deleted from the inputted text. | | | |
| **Related Use Cases:** None | | | |
| **Normal Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. The user navigates to the sentence they would like to modify. | | 1. N/A | |
| 1. The user presses delete as desired and modifies the sentence | | 1. Removes the deleted words and modifies the translated and displayed output | |
| **Alternate/Exceptional Flow of Events** | | | |
| **Actor action** | | **System Response** | |
| **None** | | **None** | |

**Predictive Typing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Predictive Typing | **ID: 5** | | **Importance:** High |
| **Primary Actor:** User | | | |
| 1. **Brief Description:** When a user inputs a phrase to be translated, their historical information is used for predictive typing | | | |
| **Preconditions:** None | | | |
| **Postconditions:** User is able to input sentences with the aid of predictions | | | |
| **Related Use Cases:** None | | | |
| **Normal Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. The user navigates to the input text box and types a word | | 1. A set of words that the user could input based on predictions is displayed on the words panel | |
| 1. The user selects/inputs the next word until a complete sentence is formed | | 1. Displays predicted possible inputs | |
| **Alternate/Exceptional Flow of Events** | | | |
| **Actor action** | | **System Response** | |
| **None** | | **None** | |

**View Translated Sentences using collapsible panels**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** View translation | **ID: 6** | | **Importance:** High |
| **Primary Actor:** User | | | |
| **Brief Description:** The user can view the translated output of their inputted text in all languages available or in their chosen output language | | | |
| **Preconditions:**   1. The user needs to have inputted a complete sentence to be translated | | | |
| **Postconditions:**   1. The translated text is displayed to the user | | | |
| **Related Use Cases:**   1. Collapse panel 2. Expand Panel | | | |
| **Normal Flow of Events** | | | |
| **Actor Action** | | **System Response** | |
| 1. The user navigates to the translated output | | 1. Display panel for translated output in chosen output languages | |
| 1. Expands panel to view translations | | 1. Displays all translated text | |
| 1. [Extend] Collapse panel to close view of translations | | 1. View is closed | |
| **Alternate/Exceptional Flow of Events** | | | |
| **Flow of Events** | | **System Response** | |
| 1. User is unable to view any translation | | 1. Error message informing user of failed translation | |

## **Nonfunctional requirements**

These are requirements that outline the operational qualities of the application and are different from the functional requirements which outline the behaviour and functionality of the product.

|  |  |
| --- | --- |
| Requirement | Requirement Description |
| Application Launch | * When the user launches the app, it needs to have an overall start-up time of less than 5 seconds, making the application responsive and available for use. |
| Uploading a PGF file | * When a user presses the upload button, they are to be directed to their device local storage and are restricted to only choosing a .PGF file for uploading. * A user file should not take longer than 5 seconds to be loaded onto the app * Once uploaded, the contents of the PGF file are to be immediately displayed on the panel |
| Changing menu options | * When a user clicks the drop-down menu on any of the menu components, the various options should appear immediately without any delay and the user should be able to switch between options seamlessly |
| Viewing Translations | * When the user clicks on the panels to view the translations, the translations should appear without any delay and without cutting out some of the information in the case where a user selects |
| Error Messages | * In the case where a user encounters errors, such as inputting an incomplete/undecipherable sentence that cannot be translated, an appropriate error message is to be displayed |
| Scalability | * The system must be scalable enough to support a .PGF file with a size of up to **100 MB** |

## **Usability Requirements**

The application’s usability is modelled using Norman’s Principles of Design. These principles are vital to observe in designing user interfaces as they allow a seamless, intuitive and user-centered experience. These requirements are outlined below:

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Requirement Description** | **Principle of Design referenced** |
| Intuitive and easy to use interface | The user interface is to make use of elements and icons (e.g., a universal icon for collapsing a panel) that users are familiar with to enable easy use | Visibility |
| Error handling | The application should not be error prone and should provide clear feedback in case an error is encountered | Feedback |
| Clarity on functionality | The function of different features/components should be made clear either by appearance or by instructions using labels. This also limits the number of points of error that a user may encounter | Affordance and Mapping |

An additional usability requirement not encompassed by the above principles of design is **Correctness.** This means that the application should produce accurate predictions in the manner requested by the user (i.e., by using the translation categories or settings chosen by the user).

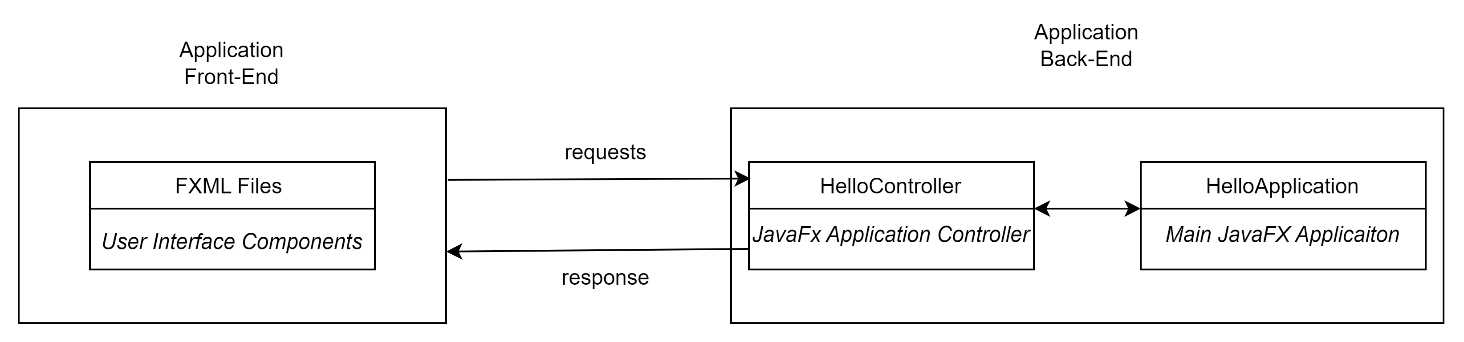
# **Design Overview**

The final product is expected to make use of the already existing Grammatical Framework Minibar code to introduce enhanced functionality of the application for the user. The final product’s user interface is to be designed using JavaFX and Scene Builder for implementation.

## **Discussion of Analysis Artefacts Produced**

During the lifecycle of the project, various analysis artefacts were produced with the aim to clearly define and communicate the system’s architecture to all parties involved. These artefacts served as a communication medium and maintained common understanding among the team members while providing a conducive plan for the application’s implementation. These artefacts are represented and discussed below:

**System Architecture Diagram**



*Figure 1: System Architecture diagram for the Grammatical Framework Minibar Application*

The above diagram represents an overview of the architecture design of the Minibar JavaFX application, using an MVC architecture pattern, represented by a Front-End and Back-end layer.

## **Architecture Pattern**

The implementation of our web application follows the Model-View-Controller development pattern. The reason we have chosen to use this model is because we used software that allowed us to separate our application into three main logical components: the model, the view and the controller, making it easy to implement this pattern. The MVC pattern enabled a well-organized and structured application with easy maintenance and modifiability, making the development process smoother and more efficient.

## **Architecture Overview**

*Front End*

The front-end of the application includes a single FXML file produced using Scene Builder to implement the User Interface of the application. This file outlines the layout of the application and is the user’s main form of interaction with the overall application. The front-end interacts with the rest of the application through interaction/data transfer with the backend, via the Application Controller as an entry point.

*Back End*

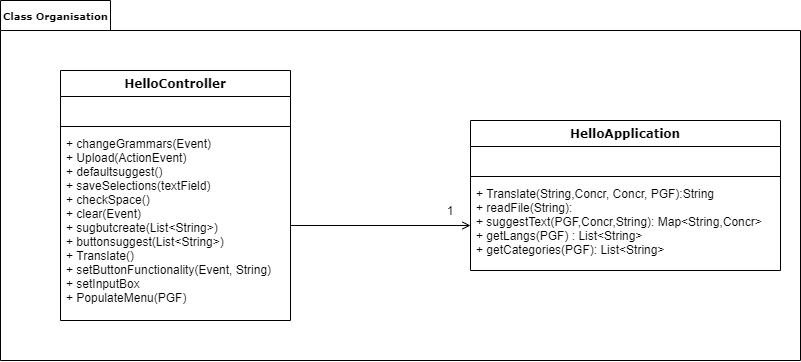
The JavaFX Application Controller (HelloController) is the main point of communication between the main logic of the application and the user interface. The controller allows for user interface management, event handling and interaction with the main logic, entry point and core of the application (HelloApplication)

## **Algorithms and Data Organization**

The implementation of the MCV pattern is the result of using Object Oriented Data Modelling to represent the data and its relationships. The use of Object-Oriented Programming allowed for code reusability, scalability and efficiency. This means we made use of classes (*figure 2*), objects, methods and attributes to structure our data using the Java programming language. The various classes that we used made use of the principles of object-oriented programming. The structure and implementation of these principles is explained in the [Implementation](#_Implementation) section.

## **Design Class Diagram**

The following Design Class Diagram offers a more in-depth overview of the structure of the application, outlining the relationships and organization of the various classes and their related attributes. The class diagram offers insight on how the application classes operate together to achieve the desired functionality. The classes depicted in the diagram show the implementation of the MCV architecture in our system. The HelloController class serves as the Controller component of the system whereas the HelloApplication class serves as the model. The view component of the system has been omitted from the class diagram as it is not a class but rather a .fxml file.



*Figure 2: Class Hierarchy diagram for the Grammatical Framework Minibar Application*

Through analysing the various application requirements and outlining these artefacts, the application was effectively developed.

# **Implementation**

To implement the functionality and features of the desktop app, we used the JavaFX library which allows developers to “design, create test, debug, and deploy rich client applications that operate consistently across diverse platforms”. Because it is written as a Java API, JavaFX application code can reference APIs from any Java library which makes it easier for us to implement our solution since we are familiar with Java. We made use of JavaFX Scene builder to interactively design the user interface. This would be used to create an enhanced version of the current UI of the minibar.

With the use of the JavaFX package, the desktop app was split into separate components to implement the features and functionality of the app. The components included the HelloApplication and HelloController classes and a hello\_view.fxml which is a file for the UI. The functionality and use of the methods in the two classes are described below.

**HelloApplication**

Class definition

The HelloApplication class serves as the main class of the application which extends the Application class that comes from JavaFX.

Relation to other class

The HelloApplication.java class makes calls to the classes and methods from the imported grammatical framework package. This allows the HelloApplication class to access methods required to implement app functionality i.e., reading in a PGF file to get the list of languages and categories. This HelloApplication is the main class of the application and initiates any initialisation of components.

Description of important methods within this class

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Parameters | Returns | Description |
| Translate | String, Concr, Concr, PGF | *String* | Returns translated text of input string |
| readFile | String | *empty* | Used to read the file selected by user to access file contents |
| getLangs | PGF | *Map<String, Concr>* | Get the languages associated with the selected grammar file |
| getCategories | PGF | *List<String>* | Get the categories associated with the selected grammar file. |
| suggestText | PGF,Concr,String | *List<String>* | Produce a list of words that the user can use as input from pgf file. |

**HelloController**

Class definition

The HelloController class manages the events and action taken on each element added to the FXML UI layout of our application. It serves as the bridge of interaction and communication between the Frontend (user interface) and the main application and coordinates the control and flow of data.

**Relation to other class**

The HelloController class is associated with the HelloApplication in that any action taken by the user in the view will prompt some method/call to be made to the Java file to access any information that needs to be displayed to the user/in the view.

**Description of important methods within this class.**

|  |  |  |
| --- | --- | --- |
| Method | Parameters | Description |
| changeGrammars | *Event* | When the user changes grammar file, method is used to clear any selections and resort to default |
| upload | *Actionevent* | Shows the new file open dialog and reads file into system |
| defaultsuggest | *empty* | Gets the user input suggestions when the text field is empty |
| saveSelections | *textField* | Saves the Input language, Output Language and Category that the user was using into an Array List to retrieve these selections when user returns to relevant text field. |
| clear | *Event* | Clears all input box and resets collapsible panel |
| sugbutcreate | *List<String>* | Creates array of button components that will be used to display suggested words |
| buttonsuggest | *List<String>* | Get list of suggested words and display them to user as buttons that they can click on as input. |
| Translate | *empty* | Get user input, translate and output to the translation panels. This method deals with the collapse and expansion of the translation panels. |
| SetButtonFunctionality | *Event, String* | check if a space has been typed and add word to sentence |
| populateMenu | *PGF* | Populates the Menu combo boxes with the Input language, Output language and Category relevant to the pgf file |

# **Program Validation and Verification**

## **Quality Management Plan**

In order to ensure and maintain the quality of the application, a quality management plan was compiled with the aim to identify quality objectives the team strives to realise in achieving a good quality product and to verify/test these objectives.

**Quality Objectives**

|  |  |  |
| --- | --- | --- |
| Quality Objective | Description | Testing Approach |
| The product is to be in line with the product scope | The product should meet all the requirements outlined by the client in terms of functionality | Validation Testing |
| Well defined classes and components | All classes of the application need to be fully functional and perform as described | Class Testing |

### Product needs to agree with the client requirements, Validation Testing

Validation Testing is a crucial step to ensure that the software meets the intended requirements and functions correctly within its intended environment. It primarily focuses on validating that the software fulfills its intended purpose and that it aligns with the stakeholders' expectations, i.e., the added features and functionalities. It verifies that the software functions as expected and delivers the right results. It ensures that the features and functionalities of the software work according to the defined specifications.

To pass the validation test, the application needs to be fully functional meaning it needs to meet the client requirements defined in the project scope. A Blackbox testing approach was taken to conduct these tests. Each feature/functionality was tested without consideration of the internal structure or organisation of the application but more on the inputs and outputs produced. This approach was effective as each feature was tested in consideration of the entire application functionality and the focus was fully on the behaviour of the system and whether this behaviour was in line with the requirements.

#### Testing Protocol

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case** | **Objective** | **Pre-condition** | **Steps** | **Post Conditions** | **Notes** | **Test Result** |
| Uploading Own Grammar file | A user needs to successfully upload their own grammar file | The user needs to have a grammar file (.PGF) stored on their local device | 1. Click on the upload button 2. Identify a .PGF file from the local device storage 3. Select the file and upload it onto the application | A grammar file loaded, and its contents displayed on the screen | A window to the user’s most recent directory on their device opens. Only available .PGF files were visible, and the file as successfully selected and uploaded onto the application.  Once uploaded, the contents of the file were displayed on the word bank/panel. | Function is according to requirements |
| Multiple Sentences | The application should allow receipt of multiple sentences as input and render translations | A grammar file should be uploaded onto the app | 1. Navigate to the first textbox 2. Input a sentence to translate and press translate 3. View translation of sentences in associated panel 4. Navigate to the next input box and repeat steps 2 and 3 | The user can enter multiple sentences into the available textboxes and the sentences will be translated and displayed in an associated panel | The user can navigate between input boxes, enter multiple sentences to be translated. Each sentence translation reflects in its associated panel | Function is according to requirements |
| Intuitive Deletion | User’s need to be able to intuitively and easily delete inputted words or phrases | There needs to be user input in a text box | 1. Click the backspace button | User can delete a word using the keyboard delete/backspace button | Unwanted word or phrase is deleted on clicking back space button | Function is according to requirements |
| Predictive Typing | User needs to be given predictions on the next word to type/input | None | 1. Users enter/selects a word 2. The word bank/panel is filtered and only suggested words, based on historical information, remain | A prediction based on user’s historical information appears when they input text |  | Function is according to requirements |
| Offline Functionality | To test whether the application is usable without any internet connectivity | The application needs to be available/installed on the device | 1. Launch the application 2. Implement the above discussed application features | All app features function completely with no internet access | The product performs all client requirements without internet access. The application functions on the user’s desktop and translations are rendered offline. | Function is according to requirements |

Product needs to make use of well-defined classes and methods, Class Testing

The aim is to test the various methods and state behavior of the application classes. To do so we made use of a white-box approach because it relied on having full knowledge of the internal structure/design of the code being tested.

*Unit Testing*

Unit Testing is a white-box approach to testing the functionality of objects or methods. The benefit of using unit testing is that it encourages modularity and separation of concerns. The code can be tested separately to ensure that the code makes use of high cohesion and loose coupling, where the objects have some dependence on each other with functionality having good level of independence. Unit Tests improve code quality and although it can be an extensive process, it ensures that we can detect bugs ad issues early in the process allowing quicker identification and resolution of problems when they are still less complicated and costly to fix.

## **Discussion of Results**

# **Conclusion**

The aim of the project was to create an enhanced version of the existing Grammatical Framework Minibar. This was to be done through the implementation of already-existing features while incorporating new features defined by the project client.

Various methods were used to design the proposed solution. These methods varied from using frameworks such as JavaFX Scene builder to build our application to following a Model View Controller Architecture pattern to organize the data.

In the implementation of our solution, we were able to produce a desktop application that:

* has offline functionality: the app can translate without the use of internet access.
* allows the user to translate at most 3 sentences at a time to allow for multiple sentence translations.
* allows the user to easily delete sentences and words.
* allow the user to view the translation of their sentences using collapsible panels that can be collapsed or expanded on user command.
* makes good use of predictive typing.

Moreover, the application has an improved UI that adheres to the 7 design principles. The UI follows a design that makes it easier for users to engage with the application.

Because of the above-mentioned, it can be said that the aims of the project were well met. We were able to meet the expectations of our scope and effectively implement the features and functionalities that were required by the client.

# **User Manual**

*See Group5\_UserManual.docx file attached.*

# **References**

<https://www.grammaticalframework.org/>

<https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm#:~:text=JavaFX%20is%20a%20set%20of,operate%20consistently%20across%20diverse%20platforms>.

Appendix A: Use Case Diagram

A diagram of a system

Description automatically generated