Implementation Report: Module 5 - Translation

Project: Inclusive Reading Aid for Dyslexic & Visually Impaired Users **Module:** Point 5 - Translation **Status:** Completed **Date:** 2025-11-08

1.0 Introduction

This report details the technical methodology for constructing the **Translation** module. This is the final functional module specified in the project proposal, designed to provide multilingual support.

The objective is to allow users to translate the text in the application—whether original, imported, or summarized—into a variety of other languages. This is achieved by integrating the `argos-translate` library, ensuring the translation process is non-blocking, and providing a clean user interface for language selection.

2.0 Core Technologies & Dependencies

This module introduces one major backend library and new UI components:

1. **argostranslate:** The core, open-source, offline translation library.
   - `argostranslate.package` : Used to manage, update, and install the required language models (packages).
   - `argostranslate.translate` : Used to list installed languages and perform the translation.
2. **PyQt6:**
   - `QComboBox` : Used to create the dropdown menu for language selection.
   - `QPushButton` : Used for the "Translate" action button.
   - `pyqtSignal` : A new signal (`translation_ready_signal`) was created to notify the main thread when the asynchronous language model setup is complete.
3. **threading:** Used to run the initial model download/installation and the translation itself in background threads, preventing UI freezes.

3.0 Implementation Methodology

The implementation involved adding new UI controls, setting up the translation engine in a non-blocking way, and routing the translation task through the existing threaded task manager.

3.1 Asynchronous Backend Initialization

A significant challenge is that `argos-translate` must download language models, which is a slow network operation.

1. `_init_translation()` **(Worker Thread):** A new function was created to handle all translation setup. To prevent blocking the application's startup, this function is launched in a new `threading.Thread` from the `__init__` constructor.
2. **Model Installation:** This thread calls `argostranslate.package.update_package_index()` and then checks if a predefined set of common languages (e.g., English, Spanish, French, German) are installed. If not, it calls `package.install()` to download them.

3. **Language Loading:** After installation, it loads all available languages into a dictionary (`self.installed_languages`) that maps the full language name (e.g., "Spanish") to its 2-letter code (e.g., "es").
4. **Ready Signal:** Once complete, it emits the `translation_ready_signal` .

3.2 GUI Construction & Population

1. **New Controls:** The "Text Tools" `QHBoxLayout` (from Module 4) was modified to include a `QLabel("Translate to:")` , a `QComboBox` (`self.language_combo`), and a `QPushButton("Translate")` .
2. **Dynamic Population:** The `translation_ready_signal` is connected to a new slot, `_populate_language_combo()` . This function runs on the main thread *after* the models are ready. It populates the `self.language_combo` with the language names from `self.installed_languages` and enables the dropdown and "Translate" button.

3.3 Task Manager Refactor

The `start_processing_task()` function (from Module 4) was refactored to be more generic, changing its signature to accept `*args` . This allows it to pass a variable number of arguments to the worker thread, which is necessary for translation (which needs `text` , `from_code` , and `to_code`) rather than just `text` .

3.4 Translation Logic

1. `start_translate_task()` **(Main Thread):** This slot is connected to the "Translate" button. It:
   - Gets the current text from `self.text_area` .
   - Gets the selected language name (e.g., "Spanish") from the `self.language_combo` .
   - Looks up the corresponding language code ("es") from the `self.installed_languages` dictionary.
   - Calls the refactored `start_processing_task()` , passing it the `_translate_thread` function, a dialog title, the text, a "from" code (hardcoded to "en" for simplicity), and the "to" code.
2. `_translate_thread()` **(Worker Thread):** This function receives the text and language codes. It:
   - Uses `argostranslate.translate.get_translation_from_codes()` to find the correct translation model.
   - Calls `installed_translation.translate(text)` to perform the translation.
   - Emits the `text_processed_signal` with the resulting translated text.
   - Includes `try...except` blocks to catch and report any translation errors.

4.0 Summary of Features Implemented

- [x] **Translation GUI:** A language selection dropdown and "Translate" button are integrated into the "Text Tools" panel.
- [x] **Automatic Model Installation:** The application runs a background check on startup to download and install required language models, without freezing the UI.

- [x] **Dynamic Language List:** The dropdown is populated only *after* the available languages are confirmed.
- [x] **Non-Blocking Translation:** The translation process itself runs in a worker thread and displays a "Processing..." dialog, ensuring the UI remains responsive.

5.0 Conclusion

Module 5 is fully implemented, providing the multilingual support requested in the project methodology. The integration of offline, asynchronous model installation and threaded translation provides a robust and user-friendly experience.

With the completion of this module, **all 6 points of the proposed methodology are now implemented and integrated**. The application is feature-complete and meets all specified objectives.