Implementation Report: Module 6 - Interface & Interaction Modes

Project: Inclusive Reading Aid for Dyslexic & Visually Impaired Users **Module:** Point 6 - Interface & Modes **Status:** Completed (Frontend Implementation) **Date:** 2025-11-08

1.0 Introduction

This report details the technical methodology for constructing the **Interface & Interaction Modes** module. This module forms the primary "control panel" of the application, providing the user with explicit choices for how they interact with the content.

This implementation builds directly upon the foundation established in **Module 2 (Dyslexia-Friendly Display)**. It adds the necessary GUI (Graphical User Interface) controls and the frontend logic to manage the application's state, switching between different modes and expanding the user's customization options.

2.0 Core Technologies & Dependencies

This module extends the existing PyQt6 implementation. The key new components utilized are:

- **PyQt6.QtWidgets:**
  - `QRadioButton` : Chosen to ensure mutual exclusivity for mode selection (i.e., only one mode can be active at a time).
  - `QGroupBox` : Used to visually group the mode-related radio buttons.
  - `QSlider` : New instances added for "Font Size" and "Reading Speed."
  - `QLabel` : Used to label the new sliders.

3.0 Implementation Methodology

The features of Module 6 were integrated into the existing `DyslexiaReaderApp` class by adding new widgets and connecting them to new and existing handler functions (slots).

3.1 Mode Selection Interface

A core requirement of this module is to provide distinct interaction modes.

1. **GUI Construction:** A `QGroupBox` titled "Select Mode" was created to semantically group the mode controls.
2. **Widget Selection:** Three `QRadioButton` widgets were added to the group: "Read Only," "Read with Highlights," and "Listen Only." Radio buttons are inherently suited for this, as they prevent conflicting states. "Read Only" is set as the default checked button.
3. **Layout:** The `QGroupBox` (containing the radio buttons in a `QHBoxLayout`) was added to the main `QVBoxLayout` , placing it prominently above the main text area and other controls.

3.2 Expanded Customization Controls

Module 6 specifies additional user settings for font size and reading speed.

1. **GUI Construction:** The existing `controls_layout` (a `QHBoxLayout` from Module 2) was expanded.

2. **New Sliders:**
   - **Font Size:** A new `QSlider` was added, labeled "Font Size." Its range was set from 12 to 36 (representing 12pt to 36pt font sizes).
   - **Reading Speed:** A new `QSlider` was added, labeled "Reading Speed." Its range was set from 50 to 200 (representing 50% to 200% speed).

3.3 Dynamic Interface Logic (Mode Switching)

The core logic of this module is handled by a new slot, `update_mode()` .

1. **Signal Connection:** The `toggled` signal of *all three* `QRadioButton` widgets was connected to the single `self.update_mode` function. This ensures the function runs any time the mode is changed.
2. **Handler Logic (`update_mode`):** This function determines the application's current state.
   - It uses a series of `if/elif` checks on `self.read_only_button.isChecked()` , `self.read_highlight_button.isChecked()` , etc.
   - **For "Read Only" and "Read with Highlights":** It ensures the main text area is visible by calling `self.text_area.show()` .
   - **For "Listen Only":** It implements this mode's primary UI requirement by *hiding* the main text area with `self.text_area.hide()` .

3.4 Dynamic Styling Logic (Font Size)

To make the "Font Size" slider functional, the existing `update_style()` function (from Module 2) was extended.

1. **Signal Connection:** The `valueChanged` signal of the new `font_size_slider` was connected to the `self.update_style` function.
2. **Handler Logic (`update_style`):** The function was modified to: a. Read the current `.value()` from `self.font_size_slider` . b. Incorporate this value into the CSS stylesheet string, adding a `font-size: {font_size}pt;` property.

This change integrates seamlessly with the existing styling for spacing and themes, allowing all visual controls to update the CSS in a single, efficient pass.

3.5 Placeholders for Module 3 (Text-to-Speech)

This module is now **frontend-complete but backend-pending**. The logic in `update_mode` and the `speed_slider` are stubs awaiting the TTS engine.

- The "Read with Highlights" and "Listen Only" modes will trigger TTS functions (e.g., `self.tts_engine.play()`) that do not yet exist.
- The "Reading Speed" slider (`speed_slider`) is not connected to any function. Its `valueChanged` signal will be connected to the TTS engine to control playback speed.

3.6 Packaging (Future Deployment Step)

The final requirement of Module 6 is packaging using PyInstaller. This is a non-code, post-development task. Once all modules are complete, a command (e.g., `pyinstaller --onefile --windowed main.py`) will be used to create a distributable, all-in-one executable file.

4.0 Summary of Features Implemented

- [x] **Mode-Switching UI:** A clear, mutually exclusive `QRadioButton` group allows users to select their preferred interaction mode.
- [x] **Mode-Switching Logic:** The application's UI visually adapts to the selected mode (e.g., hiding text for "Listen Only").
- [x] **Expanded Customization:** Functional "Font Size" slider added.
- [x] **GUI Framework:** All elements implemented within the existing `PyQt6` application structure.
- [x] **Future-Proofing:** UI controls (Speed Slider) and code stubs are in place, ready for integration with the Text-to-Speech module (Point 3).

5.0 Conclusion

The frontend and interface logic for Module 6 is complete. The application now has a robust "control panel" that fulfills all specified UI and customization requirements. The app's state logic is successfully implemented, and it is fully prepared for the backend integration of **Module 3 (Text-to-Speech)**, which will make the "Read with Highlights" and "Listen Only" modes fully functional.