

Implementation Report: Module 2 - Dyslexia-Friendly Frontend Display

Project: Inclusive Reading Aid for Dyslexic & Visually Impaired Users **Module:** Point 2 - Dyslexia-Friendly Display **Status:** Completed **Date:** 2025-11-08

1.0 Introduction

This report details the technical methodology for constructing the **Dyslexia-Friendly Display** module. This module is the core visual component of the application's frontend, responsible for rendering text in a highly accessible and customizable format.

The primary objective is to create a user interface (UI) that directly addresses the reading difficulties specified in the project proposal, including support for specialized fonts, adjustable spacing, and high-contrast themes. The implementation was developed using the **Python** programming language and the **PyQt6** framework.

2.0 Core Technologies & Dependencies

To achieve the required functionality, the following technical stack was selected:

1. **Python 3.x:** The core programming language for all application logic.
2. **PyQt6:** A modern, powerful set of Python bindings for the Qt 6 GUI framework. It is used to create and manage all elements of the desktop application.
 - `PyQt6.QtWidgets` : Utilized for all primary UI components (`QMainWindow` , `QTextEdit` , `QSlider` , `QPushButton` , `QLabel` , `QVBoxLayout` , `QHBoxLayout`).
 - `PyQt6.QtGui` : Utilized for font control (`QFont`).
 - `PyQt6.QtCore` : Utilized for core signals, slots, and enums (`Qt.Orientation`).
3. **OpenDyslexic Font:** A third-party font file (`.ttf` or `.otf`). A critical dependency of this module is that the **OpenDyslexic font must be installed on the host operating system** to be rendered correctly.

3.0 Implementation Methodology

The frontend display was constructed following a 5-step, component-based methodology.

3.1 Application Scaffolding

The foundation of the application is a `QMainWindow` . This object acts as the main container for all other UI elements. A central `QWidget` is set as its "central widget," and a `QVBoxLayout` (Vertical Box Layout) is applied to this widget to organize all sub-components vertically.

3.2 Text Display Widget

The primary area for text rendering is a `QTextEdit` widget. This widget was chosen for its robust support of rich text, custom fonts, and stylesheet-based customization. It is added to the main `QVBoxLayout` and set to be the largest, expanding element in the UI.

3.3 Typographical Styling

To meet the core requirement, the OpenDyslexic font is applied directly to the `QTextEdit` widget upon initialization. This is achieved by:

1. Instantiating a `QFont` object.
2. Setting its family name to `"OpenDyslexic"`.
3. Assigning a default, readable point size (e.g., `16pt`).
4. Applying this font object to the `QTextEdit` widget using its `.setFont()` method.

3.4 User Control Interface

To allow for user customization, a set of control widgets is placed at the top of the window, organized horizontally within a `QHBoxLayout`.

- **Line Spacing:** A `QSlider` (set to `Qt.Orientation.Horizontal`) is provided. Its range is set from 100 to 300, representing 100% to 300% line height.
- **Letter Spacing:** A `QSlider` is provided. Its range is set from 0 to 10, representing 0px to 10px of additional letter spacing.
- **Theme Control:** A `QPushButton` is used to toggle between high-contrast modes.

`QLabel` widgets are used to identify the sliders for a clear user experience.

3.5 Dynamic Styling with Signals and CSS

This is the most critical step for functionality. The static controls are connected to the `QTextEdit` widget's visual properties using PyQt's "signals and slots" system, with **Qt CSS (Stylesheets)** as the update mechanism.

1. Signals:

- The `valueChanged` signal of both the `line_spacing_slider` and `letter_spacing_slider` is connected to a single handler function (`update_style`).
- The `clicked` signal of the `theme_button` is connected to a separate handler function (`toggle_dark_mode`).

2. Slots (Handler Functions):

- `toggle_dark_mode()`: This function flips a boolean class variable (e.g., `self.is_dark_mode`) and then calls `update_style()`.
- `update_style()`: This central function is the "engine" of the display. It executes every time a control is changed. a. It reads the current `.value()` from both sliders. b. It checks the current state of the `self.is_dark_mode` variable. c. It programmatically **builds a CSS string** based on these values. d. It applies this CSS string to the `QTextEdit` widget using its `.setStyleSheet()` method.

Example CSS String Generated:

```
QTextEdit {
    line-height: 150%;
    letter-spacing: 2px;
    font-size: 16pt;
    font-family: 'OpenDyslexic';
    background-color: #2b2b2b; /* Dark mode example */
    color: #f0f0f0;           /* Dark mode example */
}
```

```
border: 1px solid #555;  
}
```

This method is highly efficient, as it batches all style updates into a single re-render and cleanly separates the logic (Python) from the presentation (CSS).

4.0 Summary of Features Implemented

This methodology successfully implements all requirements of **Point 2**:

- [x] **Dyslexia-Friendly Font:** `OpenDyslexic` is loaded and applied.
- [x] **Adjustable Line Spacing:** A `QSlider` dynamically controls the `line-height` CSS property.
- [x] **Adjustable Letter Spacing:** A `QSlider` dynamically controls the `letter-spacing` CSS property.
- [x] **High-Contrast Themes:** A `QPushButton` toggles a "dark mode" by dynamically setting the `background-color` and `color` CSS properties.
- [x] **GUI Framework:** The entire module is built on the `PyQt6` framework.

5.0 Conclusion & Next Steps

The frontend display module has been successfully implemented. It provides a stable, responsive, and highly customizable interface that meets all accessibility requirements outlined in Point 2.

The component is currently data-agnostic and displays only placeholder text. The immediate next step is the implementation of **Module 1 (Input Handling)**, which will provide the functions to load text from files (PDF, TXT) or screen captures into this completed display module.