Software Requirements Specification (SRS) Document for Aspiration Vision

Version: 1.1
Date: 17.03.2025
Author: Volodymyr Kozak

1. Introduction

1.1 Purpose

This document defines the software requirements for the real-time screen text translator desktop app. It is intended to provide a detailed overview of the system's functionalities, performance criteria, and design constraints. The SRS will serve as a reference for developers, testers, and future collaborators.

1.2 Scope

The project is a desktop application designed to:

- Capture and recognize on-screen text using OCR in real time.
- Translate the captured text via an integrated translation API.
- Overlay the translated text over the original screen content.
- Operate with a minimal, non-intrusive UI and support background execution on both Windows and Linux platforms.

1.3 Definitions, Acronyms, and Abbreviations

- **OCR:** Optical Character Recognition – the technology to convert images of text into machine-encoded text.
- **UI:** User Interface.
- **CI/CD:** Continuous Integration/Continuous Deployment.
- **API:** Application Programming Interface.
- **Rust:** A systems programming language focusing on performance and safety.
- **Python:** A high-level, general-purpose programming language widely used for data science, AI development and automation.

1.4 References

- Tesseract OCR Documentation
- GitHub Actions Documentation
- Rust Iced Documentation

- Rust Reqwest Documentation

1.5 Overview

This document describes the functional and non-functional requirements of the app, the system architecture, and outlines a plan for user interaction through user stories and use case diagram. Additional design details (e.g., color scheme, font families) can be found in a dedicated GitHub wiki pages.

2. Overall Description

2.1 Product Perspective

The app is a standalone desktop application that integrates multiple modules:

- **Python**-based OCR module.
- **Rust** for UI and backend logic.
- **Background** processing for real-time screen monitoring and overlay rendering.
- **Translation API** calls (DeepL, Google Translate) using Rust's HTTP library.

2.2 Product Functions

- **Screen Monitoring:** Continuously monitor screen updates with a debouncing delay to reduce redundant processing.
- **OCR Processing:** Capture selected screen areas and convert images to text.
- **Text Translation:** Integrate with external translation APIs to convert recognized text into the target language.
- **Overlay Display:** Render the translated text over the original text on-screen.
- **Minimal UI and Background Operation:** Run in a compact mode (e.g., system tray on Windows) while continuing operations in the background.

2.3 User Classes and Characteristics

- **General Users:** Individuals needing quick translations (e.g., travelers, professionals).
- **Power Users:** Multilingual professionals who require real-time, non-intrusive text translation.
- **Accessibility Users:** Users with disabilities who benefit from enhanced text visibility and language assistance.

2.4 Operating Environment

- **Platforms:** Windows 10/11 and popular Linux distributions (Arch/Debian).
- **Hardware:** Standard desktop/laptop systems.
- **Dependencies:** Requires network access for translation API calls; relies on OCR library (Tesseract).

2.5 Design and Implementation Constraints

- Real-time performance with minimal latency.
- Efficient resource utilization to allow background operation.
- Compatibility with cross-platform UI frameworks.

- Modular integration between Python (OCR) and Rust (translation and UI).

2.6 Assumptions and Dependencies

- The translation API will be reliable and support the required languages.
- Users have a stable internet connection when using the translation functionality.
- OCR accuracy is sufficient for the majority of common fonts and screen resolutions.

3. Functional Requirements

3.1 Screen Capture and OCR

- **FR1:** The system shall allow users to select a region of the screen for text capture.
- **FR2:** The OCR module shall recognize and extract text from the selected screen region.
- **FR3:** The system shall store recognized text briefly in memory for translation.
- **FR4:** A debounce mechanism to limit OCR processing during rapid screen changes.

3.2 Text Translation

- **FR5:** The system shall translate the extracted text into a user-selected target language.
- **FR6:** The system shall integrate with an external API via Rust's HTTP client (reqwest).
- **FR7:** Users shall be able to select the desired and the target language from a predefined list.

3.3 Overlay and UI

- **FR8:** The translated text shall overlay on the original screen text with adjustable font size, color, and opacity.
- **FR9:** The UI shall remain minimal, with only a small window or tray icon visible when the main window is closed.
- **FR10:** Users shall be able to access a "Settings" panel for API configuration, overlay customization, and toggling background mode.
- **FR11:** The overlay shall be unobtrusive and allow underlying applications to remain interactive.

3.4 Background Operation

- **FR12:** The application shall minimize to the system tray (Windows) or background process (Linux) without terminating operations.
- **FR13:** The application shall continue monitoring the screen and updating translations even when not in the foreground (if enabled).

4. Non-Functional Requirements

4.1 Performance

- **NFR1:** The OCR and translation process should complete within 2 seconds under typical conditions.
- **NFR2:** The system shall efficiently utilize system resources to avoid high CPU and memory consumption.

4.2 Usability

- **NFR3:** The UI should be simple, clean, and allow quick access to settings.
- **NFR4:** The overlay's default font size and color scheme shall be readable but unobtrusive.
- **NFR5:** The overlay shall be easily toggled on and off by the user.

4.3 Reliability and Availability

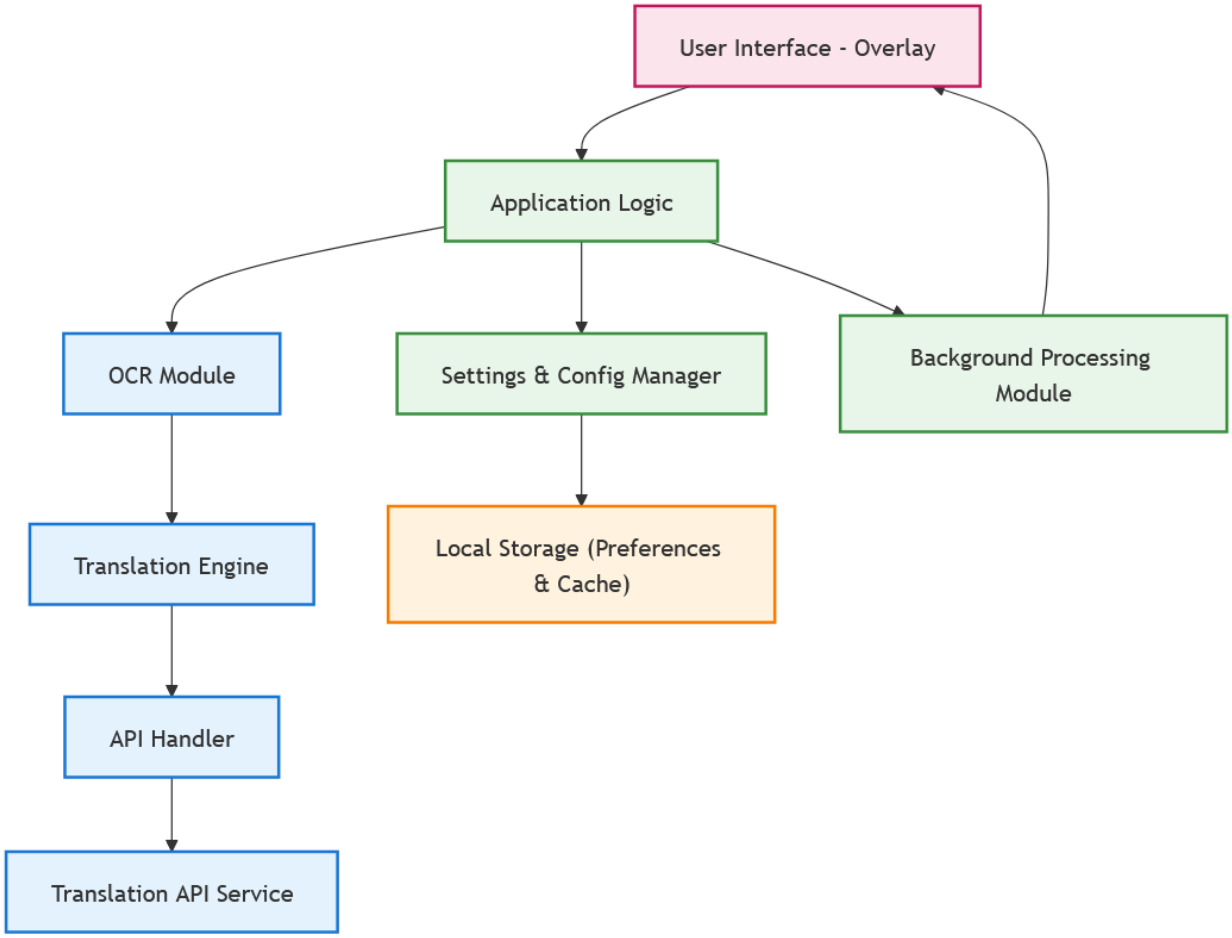
- **NFR6:** The system shall maintain continuous operation in background mode.
- **NFR7:** In case of translation failures, the system shall display an error message without crashing.

4.4 Portability

- **NFR8:** The application shall run on both Windows and major Linux distributions without significant modifications.

5. System Architecture

A high-level architectural diagram for modular design illustration:



- **User Interface - Overlay:**
  - This is the part of the application that the user directly interacts with. It includes the main window, system tray icon, and the overlay that displays the translated text on the screen.
- **Application Logic:**
  - Serves as the core coordinator. It handles incoming requests from the UI and directs them to the appropriate modules.
- **OCR Module:**
  - Captures and processes the on-screen text using Optical Character Recognition.
- **Translation Engine:**
  - Processes the text recognized by the OCR module, handling tasks such as formatting and language detection before sending it off for translation.
- **API Handler:**
  - Manages communication with external translation services.

- **Translation API Service:**
  - Represents the external translation service (such as DeepL or Google Translate).
- **Settings & Config Manager:**
  - Handles user preferences and configuration settings (API keys, language choices, overlay appearance, etc.).
- **Background Processing Module:**
  - Keeps the application running in the background, monitoring screen changes, and triggering OCR and translation processes without user intervention.
- **Local Storage (Preferences & Cache):**
  - Manages the saving of configuration data, user preferences, and possibly caching of translation results.

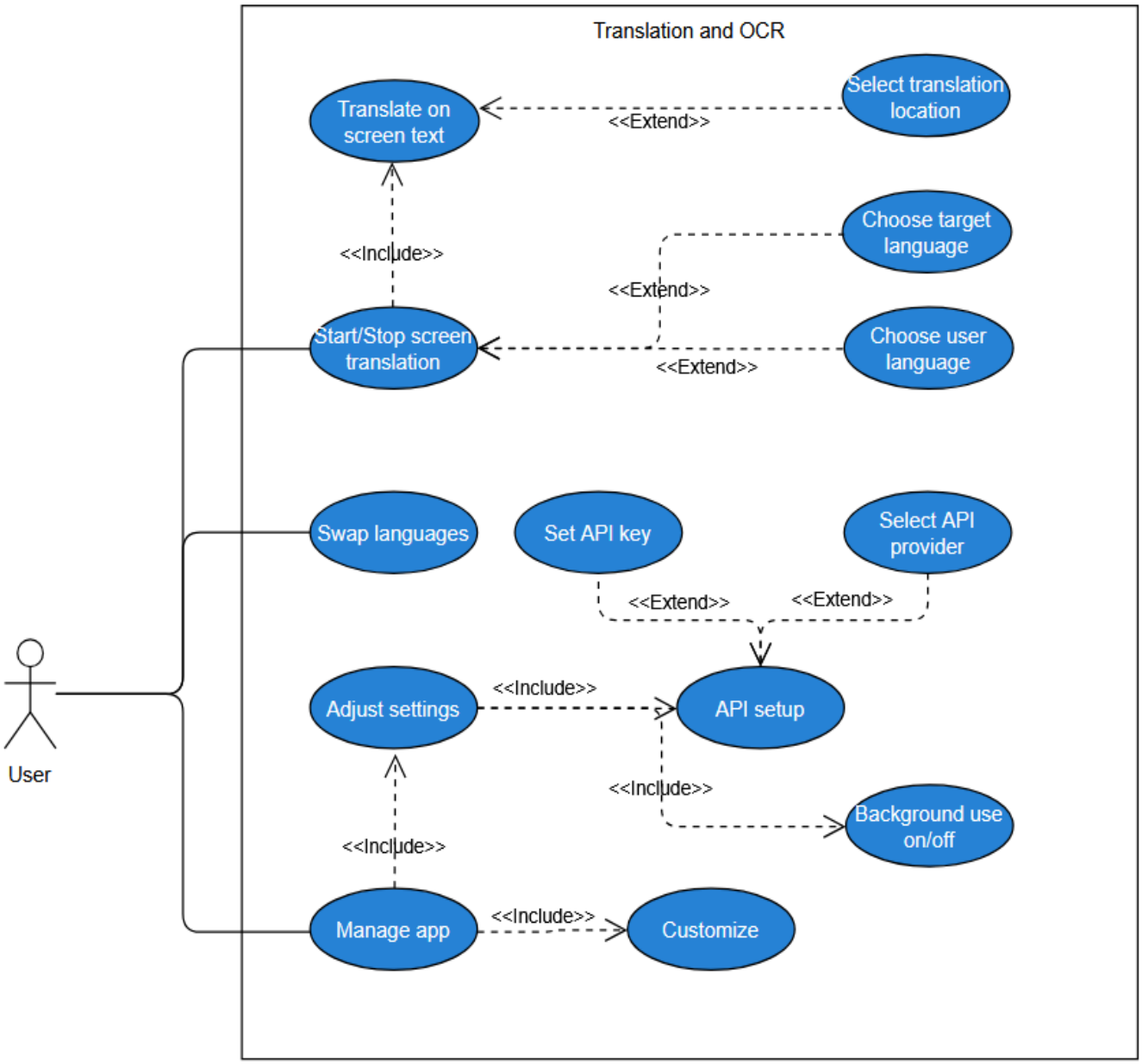
6. User Stories

1. **Screen Capture:**
   As a user, I want to select a region of my screen to capture text so that I can translate only the relevant information.
2. **Automatic OCR:**
   As a user, I want the app to automatically capture and recognize text in real time so that I do not have to manually copy text.
3. **Language Detection:**
   As a user, I want the app to detect the source language automatically so that I can quickly receive translations.
4. **Target Language Selection:**
   As a user, I want to choose a target language for translation so that I receive the output in a language I understand.
5. **Overlay Translation Display:**
   As a user, I want the translated text to be overlaid on the original text so that I can compare both versions seamlessly.
6. **Customization of Display:**
   As a user, I want to adjust the appearance of the overlay (font, size, color) to suit my preferences.
7. **API Provider Selection:**

   As a user, I want to choose the API provider that suits my preference and needs.
8. **Background Operation:**
   As a user, I want the app to run in the background so that it continues to function even when minimized.
9. **Background Operation Control:**

   As a user, I want to turn off the background operation when I don't need it to.
10. **Clipboard Integration:**
    As a user, I want to copy the translated text to my clipboard for use in other applications.
11. **Hotkey Support:**
    As a user, I want to use hotkeys to trigger screen capture and translation quickly without navigating the UI.
12. **Error Handling:**
    As a user, I want to receive notifications if the translation fails so that I can retry later.

7. Use Case Diagram



8. Other Requirements

8.1 Legal and Regulatory Requirements

- The application is intended for informational purposes only and is not a substitute for professional translation, mistakes may happen.

8.2 Data Privacy

- The application shall not store user data (preferences and customizations excluded) beyond the current session. The information from user's screen is used only to send directly to the translation API to receive a translation.

8.3 Security

- The application shall safely store user's API key using Rust keyring library.