Team 2 - Optimistic Minds
- Lakshmanan A (Developer)
- Abhishek Satish Raut (Developer)
- Nagulan M (Tester)
- Vignesh Devendran (Product Owner & Developer)

Software Requirements Specification
1. Introduction
1.1 Purpose
Selected Problem Statement:

"When another user controls the screen reader user’s machine, if they click an element, there should be a way to assign a name and shortcut to that element so the screen reader user can access it later."

Detailed Problem Statement:

When a sighted / VI user remotely controls a machine running NVDA (e.g., via Teams), clicking an element lacks a mechanism to label it with a friendly name and assign a shortcut. The NVDA user then struggles to relocate and activate it independently, hindering accessibility in collaborative scenarios like training or support.

1.2 Scope
1.2.1 In-Scope Features
- Create alternative way to Add / Remove keyboard shortcuts to input gestures for custom triggerable elements which has no default shortcuts assigned
- Both VI users & Sighted users should be able to assign shortcuts with ease
- To target for Browsers via Teams remote as priority then move on to other general apps

1.2.2 End-Users:
- Sighted Users
- NVDA - VI Users

1.3 Definitions and Acronyms
- NVDA: NonVisual Desktop Access
- OSS: Open Source Software

1.4 References
- [NVDA Add-on Scons Template](https://github.com/nvaccess/addonTemplate)
- [NVDA Add-on Development Guide](https://github.com/nvdaaddons/DevGuide/wiki/NVDA-Add-on-Development-Guide)

1.5 Overview
This SRS is organized into five main sections. Section 2 provides a high-level product perspective and functions. Section 3 details specific functional, performance, and non-functional requirements, including open source specifics like contribution guidelines. Section 4 outlines milestones, testing, and supporting information. Section 5 captures approval from representatives.

2. Overall Description
2.1 Product Perspective
Add-on extension for NVDA which extends its function to assign shortcuts to custom triggerable app specific elements.

2.1.1 List of Addons for reference
- [placeMarkers](https://github.com/nvdaes/placeMarkers)
- [gestureDuplicate](https://github.com/chaichaimee/gestureDuplicate)
- [checkGestures](https://github.com/grisov/checkGestures)

- [customLabels](https://github.com/kefaslungu/customlabels)
- [favoriteLinks](https://github.com/EdilbertoFonseca/favoriteLinks)
- [Keyboard Shortcut Autocomplete Professional for NVDA]
(https://github.com/Mohammedkhaled96/keyboard_shortcut_autocomplete_pro)

2.2 Product Functions
2.2.1 High-level features
- Select custom triggerable element in application via mouse or NVDA API
- Show dialog box pop-up to add label name and shortcut both as text or keybinding combinations
- Validate keybinding combinations
- Save entries
- List entries as standalone pop-up or in input gestures dialog box
- Edit or change entries
- Remove entries

2.3 User Classes and Characteristics
- End-users: Non-technical NVDA users
- Contributors: Developers familiar with NVDA and NVDA-addons
- Maintainers: Core team

2.4 Operating Environment
- Operating System: Windows 11
- NVDA: Add-on

2.5 Design Constraints
- License: GPL v2
- Tech stack: Python

2.6 Assumptions
- Community will self-report bugs via GitHub Issues.

3. Specific Requirements
3.1 External Interfaces
3.1.1 User Interfaces
- Custom NVDA key combinations (e.g., NVDA+Shift+V for version announcement or NVDA+A for a beep).

- Speech announcements (e.g., custom messages via ui.message() or tones via tones.beep()).

- No visual wireframes; relies on NVDA's existing dialogs or overlays for settings if needed.

3.1.2 Hardware Interfaces
- N/A

3.1.3 Software Interfaces
Add-ons hook into NVDA's Python-based architecture using modules like globalPluginHandler, appModuleHandler, and scriptHandler.

- NVDA core APIs: Events (e.g., event_gainFocus), NVDAObjects (custom overlays via chooseNVDAObjectOverlayClasses), and gestures defined in __gestures.-

- Integrations: App-specific modules (e.g., appModules/notepad.py), global plugins, or symbol dictionaries.

3.1.4 Communications Interfaces
- Managed as part of NVDA

3.2 Functional Requirements
   - Automatically detect mouse clicks during remote sessions (e.g., Teams window class or RDP detection) and capture the clicked UI element's NVDA object properties including name, role, states,

bounding rectangle, process ID, and window handle.

   - Prompt a modal, NVDA-navigable dialog for the controlling user to input a friendly name (e.g., "Submit Order Button", up to 50 characters) and select/define a shortcut (e.g., NVDA+Ctrl+1 through NVDA+Ctrl+0).

   - Store assignments in a persistent, encrypted JSON file in NVDA's user config directory, with entries including timestamp, custom name, shortcut key, and serialized element data for later validation.

3.3 Performance Requirements
- CPU Usage: Add-ons should not exceed 5-10% CPU during events; avoid long loops or heavy computations in event handlers like event_gainFocus. Use nextHandler() promptly to chain events without blocking.

- Memory Footprint: Limit to <10MB per add-on; use lazy loading (e.g., import modules only when needed) and clean up in terminate(). NVDA's Python environment is sensitive to leaks.

- Response Time: Scripts and events must execute in <100ms to prevent input lag; test on older hardware (e.g., Intel Core i5, 8GB RAM)

3.4 Non-Functional Requirements
- Security: OWASP Top 10 compliant.
- Usability: WCAG 2.1 AA accessible.
- Maintainability: modular design.
- Portability: N/A.

3.5 Open Source Specifics
- Contribution Guidelines: Link to CONTRIBUTING.md.
- Licensing: All code under [GPL v2]; docs CC-BY.
- Interoperability: Export to JSON; Import as JSON;

4. Supporting Information
4.1 Milestones
Milestone	Description	Target Date
MVP	Add-on	Mar 2026

4.2 Testing Approach
- Manual: Scratchpad, Logs
- Unit: unittest

5. Approvals
- Rep: _____ Date: ____