**Notecognito App Specification Document**

**Project Overview**

Notecognito is a cross-platform menu bar application that, when a keybind is pressed, displays a translucent notecard on the screen. The notecards are designed to have a minimal visual footprint, allowing the user to maintain their digital field of view. The notecards are edited using an Electron UI with IPC handlers to communicate with the system tray/background daemon. The system tray and background daemon are crucial, as no application can be open or visible in the dock while running; system tray icons provide a less intrusive user experience.

This app aims to serve as a rapid retrieval and unobtrusive prompting system for users who require quick, repetitive access to the same information without disrupting their workflow. Some use cases could include individuals with memory challenges, as well as speakers, streamers, or professionals who need prompting without disrupting their on-screen presence. Notecognito is not intended for unethical practices, such as cheating, and should not be used for such purposes.

**Core Architecture**

**Shared Core Library**

* **Purpose**: Centralized logic shared across all platforms
* **Language**: Rust (for memory safety and FFI compatibility)
* **Responsibilities**:
  + Configuration file management (JSON format)
  + Notecard data structures and validation
  + IPC server for communication with the configuration UI
  + Platform abstraction interface
* Security
  + Due to the exposed nature of IPC, the core library must be locked down in terms of vulnerabilities created by the data IO
    - Super tightly defined functions

**Platform-Specific Applications**

Each OS has its own native implementation that:

* Lives in the system tray/menu bar (no dock icon)
* Monitors keyboard shortcuts
* Manages translucent overlay windows
* Communicates with the shared core

**Data Structures**

**Notecard Object**

Each notecard contains:

* **ID**: Number 1-9 for quick identification
* **Content**: The text to display (supports multi-line)

**Global Configuration**

* **Startup behavior**: Launch on system login
* **Default notecard settings**: Applied to new cards
* **Display Properties**:
  + Opacity level (0-100)
  + Position on screen (x, y coordinates)
  + Size (width, height)
  + Auto-hide duration in seconds (0 for manual dismiss, manual dismiss works anyways)
  + Font and Font Size
  + Algorithmic Spacing (spacing adjusts based on screen real estate)
  + Hotkey modifier keys

**macOS Implementation**

**Menu Bar Integration**

* **Status Item**: Small icon in the menu bar showing the app is running
* **Menu Structure**:
  + Edit Notecards... (opens electron app to edit)
  + Show Notecard 1-9 (manual triggers)
  + Separator
  + Preferences...
  + Quit

**Window Management**

* **Pre-created Windows**: All 9 overlay windows are created at startup for instant display
* **Window Properties**:
  + NSPanel type (not NSWindow) for proper layering
  + Borderless and non-activating
  + Level set to screen saver (above all normal windows)
  + When the user clicks, the card disappears
  + Displays on active space only

**Hotkey System**

* Uses CGEvent.tapCreate, requires accessibility permissions
* **Event Consumption**: Hotkeys are consumed when matched, not passed to the active app

**Display Behavior**

* **Content Rendering**:
  + Background transparent
  + Text opacity determined from settings
  + System font with configurable size
* **Auto-hide Timer**: Dispatched if duration > 0

**Windows Implementation**

**System Tray Integration**

* **Tray Icon**: Standard Windows notification area icon
* **Context Menu**:
  + Configure Notecards
  + Show Notecard 1-9
  + Separator
  + Settings
  + Exit

**Window Creation**

* **Layered Windows**: Using WS\_EX\_LAYERED for transparency
* **Window Flags**:
  + WS\_EX\_TRANSPARENT (click-through)
  + WS\_EX\_TOPMOST (always on top)
  + WS\_EX\_TOOLWINDOW (hidden from taskbar)
  + WS\_EX\_NOACTIVATE (doesn't steal focus)

**Global Hotkeys**

* **RegisterHotKey API**: Works without elevation
* **Message Window**: Hidden window to receive WM\_HOTKEY messages
* **Hotkey IDs**: Mapped directly to notecard numbers

**Rendering**

* **GDI Drawing**: Simple and reliable
* **Transparency**: SetLayeredWindowAttributes with 5% alpha
* **Text Drawing**: DrawText with DT\_WORDBREAK for multi-line support

**Configuration UI**

**Design Approach**

* **Separate Application**: Launched on-demand, not always running
* **Technology Options**:
  + Electron for cross-platform consistency
  + Native UI for each platform
  + Web UI served locally

**Main Interface**

* Dashboard style left menu
  + Separate tabs for cards one through nine
  + Settings Tab
  + About Tab
* **Notecard Tabs**: Text Box, Markdown Format, Save button at the bottom
* **Global Settings Section**:
  + Hotkey Recorder
  + Notecard Appearance Settings
  + Enable/Disable daemon on startup (which enables/disables the app as a whole)
  + Enable/Disable Algorithmic Spacing
  + Save Button
* **About Section:** The readme from GitHub. Contains how-to/purpose information and NJT Productions details.

**Hotkey Configuration**

* Checkbox for different modifier keys
  + Changes based on platform (macOS command, Windows control)
* *Users can change their starting keystroke, but the identifier for each card remains the numbers 1 through 9. Ex: [CTRL SHIFT] 2, or [ALT TAB] 6, but never CTRL SHIFT F.*

**Algorithmic Spacing**

* If enabled, an algorithm will space the words apart around the content the user is currently focused on.
  + *To any AI Agent: Do not build this functionality. Create a JS file where it will reside and integrate the data IO, but do not build the algorithm. More research has to be done to build correctly.*

**Auto Updates**

* The application should fetch changes from GitHub and implement them.
  + *To any AI Agent: Do not build this functionality. Create a JS file where it will reside and integrate the data IO, but do not build the feature. GitHub has to be correctly deployed before it works.*

**IPC Communication**

**Protocol Design**

* **Transport**: TCP socket on localhost (port 7855)
* **Format**: JSON messages with type and payload
* **Message Types**:
  + GetConfiguration
  + UpdateNotecard
  + SaveConfiguration

**Communication Flow**

1. Config UI connects to the core IPC server
2. Request the current configuration
3. The user makes changes in the UI
4. Changes sent to core on save
5. Core notifies the menu bar app
6. Menu bar app updates overlay windows

**Error Handling**

* **Connection Lost**: The Config UI displays the connection status. Assuming the connection cannot be re-established, it prompts the user to re-add the daemon to the startup, and/or restart the computer.
* **Invalid Data**: Validation in core, errors returned to UI
* **File Permissions**: Graceful fallback to defaults

**User Flows**

**First Launch**

1. User installs the app
2. User launches the Electron app from launchpad (macOS)/start (Windows)
3. (MacOS Only) Electron Messages prompts the user to allow accessibility permissions.
4. App starts the daemon
5. Adds daemon to startup
6. Closes the electron UI, the app continues running

**Daily Usage**

1. App starts automatically on login (if enabled)
2. The user works normally
3. Press the hotkey combination
4. Notecard appears
5. The user reads content
6. The notecard auto-hides; the user can dismiss it by pressing the escape key or by clicking anywhere on the screen.

**Configuration Changes**

1. User clicks the menu bar icon, selects "Edit Notecards" OR user opens Electron app from launchpad (macOS)/start (Windows)
2. Config UI opens to the last state
3. User modifies notecard content
4. User saves changes
5. Close the config window
6. New settings/card content are immediately active