PRD: Instant Translator — macOS Menu Bar App

Enhanced Technical Product Requirements for Claude Code

---

1. Product Vision

Build the fastest, lowest-friction translation tool for macOS. The user never leaves their current app — they select text, press a shortcut, and the text is replaced with an intelligent translation. The app lives exclusively in the menu bar and feels like a native OS extension.

---

2. Tech Stack (Confirmed)

Component	Technology
Language	Swift 5.9+
UI Framework	SwiftUI
Target	macOS 14.0+ (Sonoma)
Build System	Swift Package Manager (SPM)
Architecture	MVVM + Services layer
LLM Provider	Anthropic Claude API (HTTP, expandable)
Persistence	UserDefaults (preferences) + Keychain (API keys)
Licensing	LemonSqueezy API (or similar, deferred)

---

3. Project Structure

```
InstantTranslator/
├── Package.swift
├── Sources/
│   └── InstantTranslator/
│       ├── App/
│       │   ├── InstantTranslatorApp.swift       # @main, NSApplication setup
│       │   └── AppDelegate.swift                # Menu bar setup, status item
│       ├── Views/
│       │   ├── MenuBarView.swift                # Main popover/panel UI
│       │   ├── StatusSection.swift              # Trial/license status + CTA
│       │   ├── PreferencesSection.swift         # Language + Tone selectors
│       │   ├── BehaviorSection.swift            # Shortcut, auto-paste, provider
│       │   ├── FooterSection.swift              # Feedback, quit, version
│       │   └── Components/
│       │       ├── ToneSelector.swift           # Segmented tab-style picker
│       │       ├── ShortcutRecorder.swift       # Custom key combo recorder
│       │       ├── APIKeyField.swift            # Secure, toggleable visibility
│       │       └── LanguageDropdown.swift       # Searchable language picker
│       ├── ViewModels/
```

```
│   │   ├── TranslatorViewModel.swift      # Core orchestration
│   │   └── SettingsViewModel.swift        # Preferences state management
│   ├── Services/
│   │   ├── TranslationService.swift       # LLM API abstraction
│   │   ├── ClaudeProvider.swift           # Anthropic API implementation
│   │   ├── OpenAIProvider.swift           # OpenAI-compatible provider (future)
│   │   ├── ClipboardService.swift         # Pasteboard read/write
│   │   ├── AccessibilityService.swift     # AX-based text selection + replacement
│   │   ├── HotkeyService.swift            # Global shortcut registration
│   │   ├── LicenseService.swift           # Trial tracking + validation
│   │   └── PromptBuilder.swift            # Constructs LLM prompts with tone/context
│   ├── Models/
│   │   ├── TranslationRequest.swift       # Input model
│   │   ├── TranslationResponse.swift      # Output model
│   │   ├── AppSettings.swift              # Persisted preferences
│   │   ├── Tone.swift                     # Enum: original, formal, casual, concise
│   │   ├── SupportedLanguage.swift        # Language enum/list
│   │   └── LicenseState.swift             # Enum: trial, active, expired
│   └── Utilities/
│       ├── KeychainHelper.swift           # Secure API key storage
│       ├── Constants.swift                # App-wide constants
│       └── Extensions/
│           └── NSEvent+Shortcut.swift     # Key event helpers
├── Resources/
│   ├── Assets.xcassets/                   # App icon, menu bar icon (template)
│   └── InstantTranslator.entitlements     # Accessibility, network entitlements
└── Tests/
    └── InstantTranslatorTests/
        ├── PromptBuilderTests.swift
        ├── TranslationServiceTests.swift
        └── LicenseServiceTests.swift
```

---

4. Core User Flow (What to Build)

4.1 The Translation Flow (Critical Path)

This is the PRIMARY feature. Everything else is secondary.

```
User selects text in ANY app
        |
        ▼
User presses ⌘⇧T (global hotkey)
        |
        ▼
App captures selected text via:
  Option A (preferred): Accessibility API (AXUIElement)
  Option B (fallback):  Simulate ⌘C → read pasteboard
        |
        ▼
App sends to LLM API:
  - Source text
  - Target language (from settings)
  - Tone (from settings)
  - System prompt (from PromptBuilder)
        |
        ▼
App receives translation
        |
        ▼
If auto-paste ON:
  - Write translation to pasteboard
  - Simulate ⌘V to replace selected text
  - Restore original pasteboard content after 2s
If auto-paste OFF:
  - Write translation to pasteboard only
  - Show brief menu bar icon animation (checkmark flash)
```

4.2 Implementation Details for Translation Flow

Global Hotkey Registration

swift

```
// Use Carbon's RegisterEventHotKey or a wrapper like HotKey (SPM package)
// Package: https://github.com/soffes/HotKey
// Default: ⌘⇧T (Cmd+Shift+T)
// Must work when app is NOT focused (global)
// Must be user-configurable via ShortcutRecorder view
```

Text Capture Strategy

swift

```
// STRATEGY 1: Accessibility API (preferred, no clipboard pollution)
// Requires: Accessibility permission granted by user
// 1. Get focused app via NSWorkspace.shared.frontmostApplication
// 2. Get AXUIElement for focused element
// 3. Read kAXSelectedTextAttribute
// 4. After translation: set kAXSelectedTextAttribute or kAXValueAttribute

// STRATEGY 2: Clipboard Simulation (fallback)
// 1. Save current pasteboard contents
// 2. Clear pasteboard
// 3. Simulate ⌘C (CGEvent)
// 4. Wait ~100ms for pasteboard to populate
// 5. Read pasteboard
// 6. [after translation] Write translation to pasteboard
// 7. Simulate ⌘V
// 8. After 2s delay, restore original pasteboard contents
```

Accessibility Permission Prompt

```
swift

// On first launch, check AXIsProcessTrusted()
// If not trusted, show a one-time onboarding panel explaining:
//   "InstantTranslator needs Accessibility access to read and replace
//    selected text in any application."
// Open System Settings → Privacy → Accessibility
// Poll AXIsProcessTrusted() every 1s until granted, then dismiss panel
```

---

5. UI Specification (Menu Bar Panel)

5.1 Panel Behavior

- **Trigger**: Click on menu bar icon (template image, 18x18pt)
- **Type**: `NSPopover` pinned to status item (not a floating window)
- **Width**: 320pt fixed
- **Height**: Dynamic, based on content (~400pt typical)
- **Dismiss**: Click outside, press Escape, or click menu bar icon again
- **Style**: `.popover` appearance with vibrancy (`.menu` material)

5.2 Panel Layout (Top to Bottom)

```
┌──────────────────────────────────────────────────┐
│ ⚠ Trial expires in 7 days          │ ← StatusSection
│ [ Buy License ]  [ Enter Key ]     │   (conditional, hide when active)
├──────────────────────────────────────────────────┤
│ Translate to:  [ English ▾ ]       │ ← PreferencesSection
│                                    │
│ Tone:                              │
│ ┌──────┬────────┬────────┬───────┐  │
│ │ OG   │ Formal │ Casual │ Brief │  │   ← Segmented control (tabs)
│ └──────┴────────┴────────┴───────┘  │
├──────────────────────────────────────────────────┤
```

```
│ Shortcut:   [ ⌘⇧T ] ← click    │ ← BehaviorSection
│             to record          │
│ Auto-paste: [====○   ] ON      │   ← Toggle switch
│                                │
│ Provider:   [ Claude ▾ ]       │
│ API Key:    [ •••••••• 👁 ]    │   ← Secure field + reveal toggle
├────────────────────────────────┤
│ 💬 Feedback        ··· ▾       │ ← FooterSection
│              ├ About           │   (three-dot menu)
│              ├ Check Updates   │
│              └ Quit            │
└────────────────────────────────┘
```

5.3 Design Language

- **Colors**: Follow system appearance (light/dark mode automatic)
- **Typography**: `.body` for labels, `.caption` for hints, SF Pro (system)
- **Spacing**: 12pt section padding, 8pt between elements within sections
- **Dividers**: Thin `Divider()` between sections
- **Accent color**: System blue for active tone tab, toggles, buttons
- **Corner radius**: 8pt for buttons, 6pt for fields
- **Tone selector**: Highlight selected tab with filled background (system accent), others have no background but text-only

---

6. LLM Integration

6.1 PromptBuilder

The prompt is the core of translation quality. The `PromptBuilder` service constructs the system + user message.

```swift
```

```swift
// PromptBuilder.swift

struct PromptBuilder {

    static func buildSystemPrompt(targetLanguage: String, tone: Tone) -> String {
        """
        You are a professional translator. Your task is to translate the user's text
        into \(targetLanguage).

        Rules:
        - Return ONLY the translated text. No explanations, no notes, no quotes.
        - Preserve the original formatting (line breaks, punctuation style).
        - If the source text is already in \(targetLanguage), improve its grammar
          and clarity instead of translating.
        \(toneInstruction(for: tone))
        """
    }

    static func toneInstruction(for tone: Tone) -> String {
        switch tone {
        case .original:
            return "- Maintain the original tone and register of the text."
        case .formal:
            return "- Use a professional, formal register. Suitable for business emails and official documents."
        case .casual:
            return "- Use a relaxed, conversational register. Suitable for chat messages and informal communication."
        case .concise:
            return "- Make the translation as brief as possible without losing meaning. Remove filler words and redundancy."
        }
    }
}
```

6.2 Anthropic Claude API Call

```swift
```

```swift
// ClaudeProvider.swift

struct ClaudeProvider: TranslationProvider {
    let apiKey: String
    let model: String = "claude-sonnet-4-20250514"  // Fast + good quality

    func translate(text: String, systemPrompt: String) async throws -> String {
        var request = URLRequest(url: URL(string: "https://api.anthropic.com/v1/messages")!)
        request.httpMethod = "POST"
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        request.setValue(apiKey, forHTTPHeaderField: "x-api-key")
        request.setValue("2023-06-01", forHTTPHeaderField: "anthropic-version")

        let body: [String: Any] = [
            "model": model,
            "max_tokens": 4096,
            "system": systemPrompt,
            "messages": [
                ["role": "user", "content": text]
            ]
        ]
        request.httpBody = try JSONSerialization.data(withJSONObject: body)

        let (data, response) = try await URLSession.shared.data(for: request)

        guard let httpResponse = response as? HTTPURLResponse,
              httpResponse.statusCode == 200 else {
            throw TranslationError.apiError
        }

        // Parse response.content[0].text
        let json = try JSONSerialization.jsonObject(with: data) as? [String: Any]
        let content = (json?["content"] as? [[String: Any]])?.first
        return content?["text"] as? String ?? ""
    }
}
```

6.3 Provider Protocol (For Extensibility)

```swift
protocol TranslationProvider {
    func translate(text: String, systemPrompt: String) async throws -> String
}
```

---

7. Data Models

7.1 Tone Enum

```swift
```

```swift
enum Tone: String, CaseIterable, Codable {
    case original = "Original"
    case formal = "Formal"
    case casual = "Casual"
    case concise = "Concise"
}
```

7.2 Supported Languages

```swift
enum SupportedLanguage: String, CaseIterable, Codable, Identifiable {
    case english = "English"
    case spanish = "Spanish"
    case french = "French"
    case german = "German"
    case portuguese = "Portuguese"
    case italian = "Italian"
    case dutch = "Dutch"
    case russian = "Russian"
    case chinese = "Chinese (Simplified)"
    case japanese = "Japanese"
    case korean = "Korean"
    case arabic = "Arabic"

    var id: String { rawValue }

    /// BCP 47 code for language detection comparison
    var bcp47: String {
        switch self {
        case .english: return "en"
        case .spanish: return "es"
        case .french: return "fr"
        case .german: return "de"
        case .portuguese: return "pt"
        case .italian: return "it"
        case .dutch: return "nl"
        case .russian: return "ru"
        case .chinese: return "zh-Hans"
        case .japanese: return "ja"
        case .korean: return "ko"
        case .arabic: return "ar"
        }
    }
}
```

7.3 App Settings (Persisted)

```swift
```

```swift
struct AppSettings: Codable {
    var targetLanguage: SupportedLanguage = .english
    var tone: Tone = .original
    var autoPaste: Bool = true
    var shortcutKeyCombo: KeyCombo = .default  // ⌘⇧T
    var selectedProvider: ProviderType = .claude
    // API key stored separately in Keychain, NOT here
}
```

7.4 License State

```swift
enum LicenseState {
    case trial(daysRemaining: Int)  // Days 1-14
    case expired                    // Day 15+
    case active(license: String)    // Valid license

    var canTranslate: Bool {
        switch self {
        case .trial: return true
        case .active: return true
        case .expired: return false
        }
    }
}
```

---

8. Services — Implementation Guide

8.1 HotkeyService

```swift
// Dependencies: HotKey package (https://github.com/soffes/HotKey)
// Add to Package.swift: .package(url: "https://github.com/soffes/HotKey", from: "0.2.1")

import HotKey

class HotkeyService: ObservableObject {
    private var hotKey: HotKey?
    var onTrigger: (() -> Void)?

    func register(combo: KeyCombo) {
        hotKey = HotKey(key: combo.key, modifiers: combo.modifiers)
        hotKey?.keyDownHandler = { [weak self] in
            self?.onTrigger?()
        }
    }

    func unregister() {
        hotKey = nil
    }
}
```

8.2 AccessibilityService

```swift
class AccessibilityService {

    /// Check if accessibility permission is granted
    static var isTrusted: Bool {
        AXIsProcessTrusted()
    }

    /// Request accessibility permission (opens System Settings)
    static func requestPermission() {
        let options = [kAXTrustedCheckOptionPrompt.takeRetainedValue(): true] as CFDictionary
        AXIsProcessTrustedWithOptions(options)
    }

    /// Get selected text from the frontmost application
    func getSelectedText() -> String? {
        guard let app = NSWorkspace.shared.frontmostApplication else { return nil }
        let axApp = AXUIElementCreateApplication(app.processIdentifier)

        var focusedElement: AnyObject?
        AXUIElementCopyAttributeValue(axApp, kAXFocusedUIElementAttribute as CFString, &focusedElement)

        guard let element = focusedElement else { return nil }

        var selectedText: AnyObject?
        AXUIElementCopyAttributeValue(element as! AXUIElement, kAXSelectedTextAttribute as CFString, &selectedText)

        return selectedText as? String
    }

    /// Replace selected text (set the selected text attribute)
    func replaceSelectedText(with newText: String) -> Bool {
        guard let app = NSWorkspace.shared.frontmostApplication else { return false }
        let axApp = AXUIElementCreateApplication(app.processIdentifier)

        var focusedElement: AnyObject?
        AXUIElementCopyAttributeValue(axApp, kAXFocusedUIElementAttribute as CFString, &focusedElement)

        guard let element = focusedElement else { return false }

        let result = AXUIElementSetAttributeValue(
            element as! AXUIElement,
            kAXSelectedTextAttribute as CFString,
            newText as CFTypeRef
        )
        return result == .success
    }
}
```

8.3 ClipboardService (Fallback)

```swift
```

```
class ClipboardService {
    private var savedContents: [NSPasteboard.PasteboardType: Data] = [:]

    func saveAndClear() {
        let pasteboard = NSPasteboard.general
        savedContents = [:]
        for item in pasteboard.pasteboardItems ?? [] {
            for type in item.types {
                if let data = item.data(forType: type) {
                    savedContents[type] = data
                }
            }
        }
        pasteboard.clearContents()
    }

    func simulateCopy() {
        // Post ⌘C via CGEvent
        let source = CGEventSource(stateID: .combinedSessionState)
        let keyDown = CGEvent(keyboardEventSource: source, virtualKey: 0x08, keyDown: true) // 'c'
        keyDown?.flags = .maskCommand
        let keyUp = CGEvent(keyboardEventSource: source, virtualKey: 0x08, keyDown: false)
        keyUp?.flags = .maskCommand
        keyDown?.post(tap: .cghidEventTap)
        keyUp?.post(tap: .cghidEventTap)
    }

    func simulatePaste() {
        let source = CGEventSource(stateID: .combinedSessionState)
        let keyDown = CGEvent(keyboardEventSource: source, virtualKey: 0x09, keyDown: true) // 'v'
        keyDown?.flags = .maskCommand
        let keyUp = CGEvent(keyboardEventSource: source, virtualKey: 0x09, keyDown: false)
        keyUp?.flags = .maskCommand
        keyDown?.post(tap: .cghidEventTap)
        keyUp?.post(tap: .cghidEventTap)
    }

    func writeText(_ text: String) {
        NSPasteboard.general.clearContents()
        NSPasteboard.general.setString(text, forType: .string)
    }

    func restoreSaved() {
        let pasteboard = NSPasteboard.general
        pasteboard.clearContents()
        let item = NSPasteboardItem()
        for (type, data) in savedContents {
            item.setData(data, forType: type)
        }
        pasteboard.writeObjects([item])
    }
}
```

8.4 LicenseService

swift

```swift
class LicenseService: ObservableObject {
    @Published var state: LicenseState = .trial(daysRemaining: 14)

    private let installDateKey = "installDate"
    private let licenseKey = "activeLicense"
    private let trialDays = 14

    init() {
        loadState()
    }

    func loadState() {
        // Check for active license first
        if let license = UserDefaults.standard.string(forKey: licenseKey) {
            state = .active(license: license)
            return
        }

        // Check trial status
        let installDate: Date
        if let saved = UserDefaults.standard.object(forKey: installDateKey) as? Date {
            installDate = saved
        } else {
            installDate = Date()
            UserDefaults.standard.set(installDate, forKey: installDateKey)
        }

        let daysSinceInstall = Calendar.current.dateComponents([.day], from: installDate, to: Date()).day ?? 0
        let remaining = max(0, trialDays - daysSinceInstall)

        if remaining > 0 {
            state = .trial(daysRemaining: remaining)
        } else {
            state = .expired
        }
    }

    func activateLicense(_ key: String) async -> Bool {
        // TODO: Validate with LemonSqueezy API
        // For now, accept any non-empty key
        UserDefaults.standard.set(key, forKey: licenseKey)
        state = .active(license: key)
        return true
    }
}
```

---

9. TranslatorViewModel (Core Orchestrator)

swift

```swift
@MainActor
class TranslatorViewModel: ObservableObject {
    @Published var settings: AppSettings
    @Published var isTranslating: Bool = false
    @Published var lastError: String?

    private let accessibilityService = AccessibilityService()
    private let clipboardService = ClipboardService()
    private let hotkeyService = HotkeyService()
    private let licenseService: LicenseService
    private var translationProvider: TranslationProvider

    init(licenseService: LicenseService) {
        self.licenseService = licenseService
        self.settings = AppSettings.load()  // From UserDefaults
        self.translationProvider = Self.makeProvider(for: settings)

        setupHotkey()
    }

    private func setupHotkey() {
        hotkeyService.onTrigger = { [weak self] in
            Task { @MainActor in
                await self?.performTranslation()
            }
        }
        hotkeyService.register(combo: settings.shortcutKeyCombo)
    }

    func performTranslation() async {
        // 1. Check license
        guard licenseService.state.canTranslate else {
            // Bounce menu bar icon to draw attention
            animateMenuBarIcon()
            return
        }

        // 2. Get selected text
        var sourceText: String?

        if AccessibilityService.isTrusted {
            sourceText = accessibilityService.getSelectedText()
        }

        // Fallback to clipboard if AX failed
        if sourceText == nil || sourceText?.isEmpty == true {
            clipboardService.saveAndClear()
            clipboardService.simulateCopy()
            try? await Task.sleep(nanoseconds: 150_000_000) // 150ms
            sourceText = NSPasteboard.general.string(forType: .string)
        }

        guard let text = sourceText, !text.isEmpty else {
            lastError = "No text selected"
            return
        }
```

```
        // 3. Build prompt
        let systemPrompt = PromptBuilder.buildSystemPrompt(
            targetLanguage: settings.targetLanguage.rawValue,
            tone: settings.tone
        )

        // 4. Translate
        isTranslating = true
        defer { isTranslating = false }

        do {
            let translated = try await translationProvider.translate(
                text: text,
                systemPrompt: systemPrompt
            )

            // 5. Output
            if settings.autoPaste {
                // Try AX replacement first
                if AccessibilityService.isTrusted,
                   accessibilityService.replaceSelectedText(with: translated) {
                    // Success via AX — no clipboard pollution
                    return
                }

                // Fallback: clipboard paste
                clipboardService.writeText(translated)
                clipboardService.simulatePaste()

                // Restore clipboard after delay
                Task {
                    try? await Task.sleep(nanoseconds: 2_000_000_000) // 2s
                    clipboardService.restoreSaved()
                }
            } else {
                // Just copy to clipboard
                clipboardService.writeText(translated)
                flashMenuBarIcon()  // Brief checkmark animation
            }
        } catch {
            lastError = error.localizedDescription
        }
    }
}
```

---

10. App Entry Point & Menu Bar Setup

swift

```swift
// InstantTranslatorApp.swift
import SwiftUI

@main
struct InstantTranslatorApp: App {
    @NSApplicationDelegateAdaptor(AppDelegate.self) var appDelegate

    var body: some Scene {
        // No WindowGroup — this is a menu bar-only app
        Settings { EmptyView() }
    }
}

// AppDelegate.swift
import SwiftUI

class AppDelegate: NSObject, NSApplicationDelegate {
    var statusItem: NSStatusItem!
    var popover: NSPopover!
    let licenseService = LicenseService()
    lazy var viewModel = TranslatorViewModel(licenseService: licenseService)

    func applicationDidFinishLaunching(_ notification: Notification) {
        // Create status bar item
        statusItem = NSStatusBar.system.statusItem(withLength: NSStatusItem.squareLength)

        if let button = statusItem.button {
            button.image = NSImage(systemSymbolName: "text.bubble", accessibilityDescription: "InstantTranslator")
            button.action = #selector(togglePopover)
        }

        // Create popover
        popover = NSPopover()
        popover.contentSize = NSSize(width: 320, height: 420)
        popover.behavior = .transient  // Dismiss on click outside
        popover.contentViewController = NSHostingController(
            rootView: MenuBarView(viewModel: viewModel, licenseService: licenseService)
        )

        // Hide dock icon (menu bar app only)
        NSApp.setActivationPolicy(.accessory)

        // Check accessibility on launch
        if !AccessibilityService.isTrusted {
            AccessibilityService.requestPermission()
        }
    }

    @objc func togglePopover() {
        guard let button = statusItem.button else { return }
        if popover.isShown {
            popover.performClose(nil)
        } else {
            popover.show(relativeTo: button.bounds, of: button, preferredEdge: .minY)
        }
```

```
    }
}
```

---

11. Package.swift

```swift
// swift-tools-version: 5.9

import PackageDescription

let package = Package(
    name: "InstantTranslator",
    platforms: [.macOS(.v14)],
    dependencies: [
        .package(url: "https://github.com/soffes/HotKey", from: "0.2.1"),
    ],
    targets: [
        .executableTarget(
            name: "InstantTranslator",
            dependencies: ["HotKey"],
            path: "Sources/InstantTranslator"
        ),
        .testTarget(
            name: "InstantTranslatorTests",
            dependencies: ["InstantTranslator"],
            path: "Tests/InstantTranslatorTests"
        ),
    ]
)
```

---

12. Entitlements Required

```xml
<!-- InstantTranslator.entitlements -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "...">
<plist version="1.0">
<dict>
    <!-- Network access for API calls -->
    <key>com.apple.security.network.client</key>
    <true/>
    <!-- Accessibility API access -->
    <key>com.apple.security.accessibility</key>
    <true/>
    <!-- App Sandbox (for distribution) -->
    <key>com.apple.security.app-sandbox</key>
    <true/>
</dict>
</plist>
```

> **Note**: If sandboxed, Accessibility API via AXUIElement requires a **Temporary Exception entitlement** or the app must be distributed outside the Mac App Store. For initial development, build **without sandbox** and use code signing + notarization for direct distribution.

---

13. Build Order (Phases for Claude Code)

Phase 1: Skeleton + Menu Bar (Do this first)

1. Set up `Package.swift` with HotKey dependency
2. Create `InstantTranslatorApp.swift` + `AppDelegate.swift`
3. Get a status bar icon showing with an empty popover
4. Verify app runs as menu-bar-only (no dock icon)

Phase 2: UI Panel

1. Build `MenuBarView.swift` with all four sections (static data)
2. Implement `ToneSelector` as segmented control
3. Implement `LanguageDropdown` with all languages
4. Implement toggle for auto-paste
5. Wire UI to `SettingsViewModel` with `@AppStorage` persistence

Phase 3: Global Hotkey

1. Integrate HotKey package
2. Register default ⌘⇧T
3. Build `ShortcutRecorder` view for custom combos
4. Verify hotkey fires when app is in background

Phase 4: Text Capture + Replacement

1. Implement `AccessibilityService` (get + replace selected text)
2. Implement `ClipboardService` as fallback
3. Add accessibility permission check + prompt flow
4. Test in Safari, Notes, Slack, VS Code

Phase 5: LLM Translation

1. Implement `PromptBuilder`
2. Implement `ClaudeProvider` with HTTP call
3. Implement `TranslationService` (provider abstraction)
4. Wire `TranslatorViewModel.performTranslation()`
5. Test end-to-end: select → shortcut → translated text appears

Phase 6: Polish & Licensing

1. Implement `LicenseService` (trial countdown)
2. Build `StatusSection` UI (trial banner, buy/enter key)
3. Add `APIKeyField` with Keychain storage

4. Add menu bar icon animations (loading spinner, success checkmark)
5. Add error handling (no selection, API error, network timeout)
6. Add "Feedback" and "Quit" in footer

---

14. Error Handling Matrix

Error	User Experience	Implementation
No text selected	Menu bar icon briefly shakes	Check for empty string after capture
API key missing	Open popover, highlight API Key field	Check before API call
API rate limit	Menu bar tooltip: "Rate limited, retry in Xs"	Parse 429 response, retry with backoff
Network offline	Menu bar tooltip: "No connection"	URLSession error handling
Translation failed	Menu bar tooltip: "Translation failed"	Generic catch, log error
Trial expired	Open popover showing "Trial Expired" banner	LicenseService blocks translation
Accessibility denied	Show one-time prompt to enable	AXIsProcessTrusted() check

---

15. Testing Checklist

- [ ] App launches in menu bar only (no dock icon)
- [ ] Popover opens/closes on status item click
- [ ] Popover dismisses when clicking outside
- [ ] Language selection persists after quit/relaunch
- [ ] Tone selection persists after quit/relaunch
- [ ] Global hotkey works when app is NOT focused
- [ ] Selected text is captured from: Safari, Notes, TextEdit, Slack, VS Code
- [ ] Translation replaces selected text (auto-paste ON)
- [ ] Translation copies to clipboard only (auto-paste OFF)
- [ ] Original clipboard restored after auto-paste
- [ ] API key stored securely in Keychain
- [ ] API key survives app restart
- [ ] Trial countdown works correctly
- [ ] Expired trial blocks translation
- [ ] Light/dark mode renders correctly
- [ ] Custom shortcut recording works
- [ ] App handles no internet gracefully

---

16. Out of Scope (v1)

- Multiple simultaneous providers
- Translation history/log
- Auto-detect source language toggle in UI (LLM handles this implicitly)
- Batch/bulk translation

- Document translation (files)
- iOS/iPadOS version
- Mac App Store distribution (use direct + notarization)
- Onboarding wizard (keep it minimal — just the AX permission prompt)