ChatGPT

Ingenious Translate – Product Requirements Document

Title Page

- **Product Name:** Ingenious Translate
- **Version:** 1.0
- **Date:** 2026-02-04
- **Author(s):** Ingenious Translate Product Team

Table of Contents

Executive Summary

The purpose of this Product Requirements Document is to define the features, functionality, and strategy for **Ingenious Translate**, aligning stakeholders and guiding development. It outlines what capabilities must be built into the first release and beyond [1]. Ingenious Translate is a cross-platform translation application that leverages on-device AI models for **real-time text and speech translation**. It is designed to support multiple languages and to work even without an internet connection. By using modern ML runtimes (like ONNX and transformers.js) and caching mechanisms, it achieves **offline functionality** and strong data privacy [2] [3]. The global market for translation apps is growing rapidly, projected from ~$3.5 B in 2023 to $9.8 B by 2032 [4], driven by increased globalization, travel, and AI advancements [4] [5]. Ingenious Translate will address this demand by providing accurate, accessible, and private translation services.

Vision and Strategy

Vision Statement

Ingenious Translate's vision is to **break down language barriers** by empowering people around the world to communicate effortlessly, regardless of language. We aim to deliver a user-friendly, AI-powered translator that **"bridges the language gap"** for travelers, businesses, and learners [6]. The product will focus on privacy and offline use, ensuring that sensitive conversations and content **never leave the user's device** [7] [8]. This aligns with scenarios such as legal or medical transcription, where confidentiality is critical [8].

Goals and Objectives

- **High-Quality Translation:** Provide accurate translation for dozens of major language pairs. Utilize state-of-the-art ONNX models (e.g., OPUS-MT) converted to int8 for smaller size and faster inference [9].

- **Real-Time Speech Support:** Enable real-time speech recognition, translation, and speech synthesis so that spoken conversations can be translated fluidly. Support continuous or push-to-talk modes with minimal latency.
- **Offline Capability:** Ensure the app can operate fully offline after initial model downloads. Cache models and data locally (IndexedDB) so that translations can be done without network access [2] [3].
- **Privacy and Security:** Keep all translation and speech data on-device. No audio or text should be sent to external servers, protecting user privacy [7] [8].
- **Performance and UX:** Achieve fast response times (e.g., multi-language model loading under a few seconds and translation responses in real-time). Design a clear, intuitive UI (responsive for mobile) to simplify translations (copy/clear buttons, progress indicators) [10] [11].
- **Accessibility:** Build an inclusive product, e.g. by providing text outputs for speech (for the hearing impaired) and supporting screen readers and high-contrast modes.

Target Users and Personas

- **International Travelers:** People who frequently travel or live abroad and need quick translations in areas with limited internet. A traveler in Japan, for example, may use Ingenious Translate offline to communicate with locals. The growing travel industry has **augmented demand for translation tools** [6].
- **Business Professionals:** Sales, support, or management staff who engage in global meetings. They need reliable translation for documents and speech to coordinate with international partners. For them, accuracy and confidentiality are crucial in sensitive contexts [8] [6].
- **Language Learners and Students:** Users studying foreign languages will use the app to understand and pronounce phrases. They benefit from seeing and hearing translations, improving vocabulary and comprehension.
- **Content Creators / Educators:** Creators who publish multilingual content or educators teaching language can use the app to generate translations and accurate pronunciations, even offline.
- **Developers and Technologists:** As an advanced tech demo, developers interested in AI/ML for NLP will appreciate open models and web technologies that Ingenious Translate offers.

User Stories

- **As a Traveler**, I want to speak into the app and have it translate my speech into the local language, so that I can ask for directions and understand responses without knowing the language.
- **As a Business User**, I want to paste a paragraph of text (e.g., an email) and get an accurate translation in another language, so that I can communicate with international colleagues and clients.
- **As a Commuter in an Offline Area**, I want to have translations available even without internet, so that I'm not stranded by lack of connectivity.
- **As a Language Learner**, I want to hear the pronunciation of translated text in a natural voice, so that I can learn how to speak the language correctly.
- **As a Hearing-Impaired User**, I want speech from others translated to text on my screen, so that I can participate in conversations in my own language.
- **As a Privacy-Conscious User**, I want confidence that none of my conversations or documents are sent to the cloud, so that my personal or business data remains confidential.

Functional Requirements

- **Text Translation:** Translate written text between supported languages. The system shall load a pre-trained translation model (e.g. OPUS-MT converted to ONNX) for each language pair. Once downloaded, the model is cached locally (IndexedDB) and reused for offline translation [2] [3]. For example, selecting "English → German" loads the corresponding model. The translation inference should generate output up to a set token limit (e.g., ≤512 tokens in, ≤256 tokens out) [12].
- **Speech Recognition (ASR):** Convert spoken input to text in the source language. Use on-device speech-to-text models (e.g., OpenAI Whisper or similar via WebAssembly) or the browser's native SpeechRecognition API [7] [13]. The ASR engine should support multiple languages for recognition and operate with low latency. It must run locally in a Web Worker (or use the device's built-in API) to ensure privacy and performance [14] [7].
- **Speech Translation:** Chain the speech recognition output through the translation model to perform speech-to-speech translation. The user speaks in Language A, the app recognizes the speech, translates the text into Language B, and optionally speaks the result. Real-time or near-real-time pipeline is needed. Intermediate text transcripts should be shown to the user for confirmation or correction. The system should handle push-to-talk or continuous modes.
- **Text-to-Speech (TTS):** Generate audio output for translated text. Utilize the Web Speech API's SpeechSynthesis for supported languages and voices [15]. Provide natural-sounding TTS with selectable voice parameters (gender, pitch, speed) if available. The TTS should accurately pronounce the translated text in the target language. If browser support is limited, allow integration with optional offline TTS models or third-party services as a fallback.
- **Model Management and Caching:** On first use of a language model, download the ONNX model files from the backend or a model repository. Store these files in the browser's IndexedDB to avoid re-downloading [3]. The app should automatically use cached models on subsequent loads (even when offline) [2]. If a newer model version is available, the app should detect and update it. Provide UI controls to manage models (e.g., select or remove language pairs).
- **UI Components:** Provide a clean, responsive interface. Key UI components include:
- **Language Selection:** Dropdowns or buttons to choose source and target languages.
- **Text Input/Output:** A text area for the user to type or paste input, and a read-only text area for the translated output [16].
- **Control Buttons:** "Load Model" (to fetch required ML model), "Translate" (to perform translation), "Copy" (copy the translated text to clipboard), and "Clear" (reset input and output fields) [17].
- **Microphone Input:** A mic icon/button to start/stop speech recognition.
- **Speech Output:** A speaker icon to replay the translated speech.
- **Status Field:** An area to show loading progress or status messages during model loading or translation [16].
  All UI components should adapt to different screen sizes (responsive design) [18].

Non-Functional Requirements

- **Performance:** The application must be responsive and fast. Model downloads (potentially hundreds of MB) should show progress and complete in a reasonable time. Once loaded, translation inference should complete in near-real time (ideally under a second for moderate text lengths). Using quantized int8 ONNX models significantly reduces size (4–5× smaller) and speeds up inference [9]. Parallelize heavy tasks using Web Workers to keep the UI thread responsive [14] [19].

- **Security and Privacy:** All processing should occur client-side. No audio or text is sent to cloud services by default, protecting user privacy [7]. Use HTTPS if any external services are accessed. Secure all stored data (IndexedDB) from unauthorized access. Comply with relevant data protection regulations. The app should prompt for microphone permission explicitly and handle refusals gracefully. In high-security scenarios (e.g. court reporting), ensure transcripts and translations remain entirely local [8] [7].
- **Accessibility:** Follow WCAG guidelines: ensure high-contrast UI, resizable text, and keyboard navigation. Provide ARIA labels for UI controls. Use clear visual and/or audio cues for status changes (not relying on color alone) [20]. All non-text content (like speech) should have text equivalents. Enable captions or transcripts for audio output to assist hearing-impaired users.
- **Offline Capability:** The product shall be fully functional offline after initial model downloads. All mandatory features (text translation, speech recognition and translation, TTS) should work without an internet connection. Cached models (IndexedDB) make this possible [2] [3]. Offline mode is essential for users in connectivity-limited environments (e.g., remote travel destinations or secure facilities [8] [3]).

UI/UX Requirements

- **Layout Overview:** The main screen is divided into panels for source and target. For example, the left side holds "Input Language," "Input Text," and speech input buttons; the right side shows "Output Language," "Translated Text," and TTS controls. Buttons are clearly labeled (e.g., "Translate", "Copy", "Clear"). The design is mobile-first and adjusts to narrow screens [18].
- **Interaction Flows:** Follow a guided flow. Example: (1) User selects languages and taps "Load Model". (2) UI shows "Loading (X%)" as the model downloads [16]. (3) Once ready, user types or speaks input. (4) User taps "Translate" – the button is disabled during processing, and status shows "Translating..." [16]. (5) Translated text appears, and the user can press "Copy" or play the TTS audio. These steps should be intuitive and have minimal clicks.
- **Progress and Status Indicators:** Show clear indicators for lengthy operations. For example, display a percentage progress bar while a model is loading [16]. Show dynamic status text ("Model loaded – ready to translate", "Translating...", etc.) so users understand what's happening [16]. Disable or gray-out controls (like the Translate button) while actions are in progress to prevent duplicate commands.
- **Error States:** Provide helpful error messages. Example error states include: model download failure ("Error loading model – please reload"), unsupported language pair, or microphone access denied. Errors should be shown near the relevant component (e.g., an alert box near the status field) and suggest next steps (e.g., "Check your connection" or "Grant microphone permission"). In offline mode, if a required model is not cached, inform the user that they need to go online first. Error messages should be concise and clear [21].

Technical Architecture

- **Frontend Stack:** Develop the UI as a progressive web app using modern JavaScript frameworks (e.g. React or Vue.js) for component structure and state management. Use CSS frameworks (like Tailwind or Bootstrap) to ensure responsive design. The app should be a single-page web app that can also be packaged as a desktop/mobile app via Electron or similar if needed.
- **ML Runtime (ONNX, Transformers.js):** Use the Hugging Face Transformers.js library with ONNX Runtime Web to run AI models in-browser. This setup uses WebAssembly (WASM) under the hood for inference [7] [22]. Example architecture: the main JS thread triggers `pipeline("translation",`

`modelId)`, which spins up an internal Web Worker that loads the ONNX model and runs inference [23]. This allows running the same transformer models directly in JS with good performance [19]. Models should be quantized to reduce size (int8) and ensure faster execution [9].

- **Speech API Integration:** Integrate the Web Speech API for baseline support. Use `SpeechRecognition` (or `webkitSpeechRecognition`) for voice input, and `SpeechSynthesis` for TTS [15]. However, since many browsers' default speech recognition is cloud-based, also include an offline ASR option (like Whisper in WASM) for privacy. The Picovoice SDK and similar libraries demonstrate using WebAssembly and Web Workers for on-device speech processing [14]. Thus, audio capture goes through a Web Worker or dedicated engine, then text results are passed back to the main app.
- **Storage Strategy (IndexedDB, LocalStorage):** Store large ML model files in IndexedDB [3]. Upon first download, save the model parts to IndexedDB so subsequent loads (even on page reload or offline) read from cache. Store user preferences (e.g. last-used languages, selected voices) in LocalStorage or IndexedDB as well. Ensure the storage scheme checks for updated model versions (e.g., using a version key) so that the app can refresh outdated models.
- **Web Worker Message Protocol:** Define a clear message contract between the main thread and worker. For example, messages like `{ type: 'loadModel', modelId: 'opus-mt-en-de' }` to load a model, and `{ type: 'translate', text: '...' }` to run inference. Workers should respond with progress updates (`{ type: 'progress', percent: 42 }`) and results (`{ type: 'result', text: '...' }`). Keeping a simple JSON protocol ensures decoupling and ease of debugging. This follows patterns seen in ONNX Runtime Web and Transformers.js where model loading and inference are isolated in a worker [23] [22].

APIs and Interfaces

- **Translation and Model APIs:** The primary translation functionality is on-device. However, architect the system to optionally fall back on external APIs if needed. For example, allow integration with cloud services (Google Translate, Microsoft Azure Translator, Hugging Face Inference API) via REST, in case the offline model is not available for a requested language. The app's configuration should include API keys (if used) stored securely. Each API integration should have an abstraction layer so that swapping providers is easy. Use asynchronous fetch calls for these services.
- **Worker Communication Contracts:** Specify the exact request/response formats for worker messages. For instance:
- **Request:** `{type: "translate", payload: {sourceLang: "en", targetLang: "de", text: "Hello"}}`
- **Response:** `{type: "translatedText", payload: {text: "Hallo"}}`
- **Error:** `{type: "error", payload: {message: "Model load failed"}}`
  Document all message types (loadModel, translateText, startASR, stopASR, etc.) so front-end and worker code agree on the protocol. This interface serves as an internal API between the UI and background processing.

Metrics and KPIs

- **Translation Accuracy:** Achieve a high level of quality (e.g., BLEU or human eval above a threshold) for supported language pairs. Track user feedback on accuracy.

- **Model Loading Time:** Average time to download and initialize a model (target: under 10 seconds on a 5G connection). Use browser performance API to measure.
- **Response Latency:** Time from input submission to translation output (target: real-time conversation speed for typical input).
- **Usage Metrics:** Number of translations performed per session, average session length, languages most used. These help prioritize support for additional languages.
- **User Engagement:** Active users, repeat usage rate, app rating (if published).
- **Error Rate:** Track frequency of failures (e.g., model load errors, recognition errors) and crashes.
- **Resource Utilization:** Monitor memory usage for running models, to ensure it stays within device constraints.

Release Plan and Milestones

- **MVP (Phase 1):** Basic text translation and speech recognition for a core set of languages (e.g., English, Spanish, French, Chinese, Hindi, Arabic). Include UI for selecting languages, input/output text, and basic TTS for one language (e.g., English). Offline model download and caching must be functional [2] [3]. Achieve core KPIs: acceptable accuracy and sub-5s model load.
- **Phase 2:** Add speech-to-speech translation in both directions with live microphone input and audio output. Implement multiple voices for TTS, improve UI (language auto-detect, history of translations). Introduce offline ASR (like Whisper) to replace or augment browser API for greater privacy [7].
- **Phase 3:** Expand language coverage (add less common languages), optimize performance (model quantization updates, WebGPU support). Implement advanced features like continuous streaming translation and background listening (wake-word activation as optional). Gather analytics data to refine models.
- **Phase 4:** Integration with external APIs as fallback, cloud sync of user preferences, and extensions (e.g., image-to-text translation in documents). Launch on mobile app stores and desktop (Electron) with polished UX. Conduct accessibility audit and ensure compliance.

Risks and Mitigations

Technical Risks

- **Model Size and Performance:** Large translation models (hundreds of MB) may take long to download or exceed device memory. *Mitigation:* Use quantized models (int8) to reduce size [9]. Provide a progress bar during load and warn users about download size. Allow selecting only needed language pairs.
- **Inference Latency:** On older devices or browsers, inference may be slow. *Mitigation:* Test on target devices; optimize by limiting input length and using web workers [14] [19]. Consider tiered models (faster but smaller vs. slower high-quality). Use WebAssembly SIMD or WebGPU if available.
- **Browser Compatibility:** Not all browsers support needed APIs (e.g., SpeechRecognition or IndexedDB quirks). *Mitigation:* Develop for evergreen browsers first, use polyfills if necessary. Document minimum requirements (e.g., recent Chrome/Edge/Firefox). Provide fallback messages if a feature isn't supported.
- **Offline Limitations:** Initial model download requires connectivity; if user lacks internet permanently, first-run use is blocked. *Mitigation:* Warn user to connect at least once. Possibly distribute common models in the app package for critical languages.

Product-Level Risks

- **Market Competition:** Major players (e.g., Google Translate) exist. *Mitigation:* Differentiate on privacy (fully offline) and speed. Highlight unique features (e.g., no data sent to cloud).
- **User Adoption:** Complex AI features may confuse some users. *Mitigation:* Invest in a very simple UI/UX (guided flow, tooltips). Validate with user testing and iterate.
- **Legal/Privacy:** Storing and processing speech/text data could raise compliance issues. *Mitigation:* By design, no user data leaves the device. Open-source the client code for transparency.
- **Language Coverage:** Unable to cover all languages at launch. *Mitigation:* Clearly publish supported languages and roadmap for new ones. Allow community contributions for model expansion.

Open Questions

- Which exact models and languages should be prioritized first? How will we measure “acceptable accuracy” for each language?
- Should we build our own models or rely exclusively on open ones (e.g., Hugging Face OPUS-MT)? What are the licensing implications?
- How will we handle dialects and non-Latin scripts?
- What are the policy and business rules for fallback to cloud APIs (e.g., usage limits, privacy terms)?
- How to monetize (if at all)? Will the app be free, freemium, or ad-supported?
- What metrics will be tracked internally, and how to handle user analytics ethically (opt-in)?

Approvals and Review Sign-offs

- **Product:** ____ (Name/Title) – Date: __
- **Engineering:** ____ **(Name/Title) – Date:** _____
- **Design:** ____ *(Name/Title) – Date:* _
- **Legal:** _____ (Name/Title) – Date: __
- **Quality Assurance (QA):** ___ *(Name/Title) – Date:* _

Sources: Market and technology trends are cited above [4] [7] [2] [3] [16].

---

[1] What is a Product Requirements Document (PRD)? | Atlassian
https://www.atlassian.com/agile/product-management/requirements

[2] [9] [10] [11] [12] [16] [17] [18] [21] [23] GitHub - harisnae/multilingual-translator-offline: Offline multilingual translator using ONNX-quantised OPUS-MT models run entirely in the browser via @xenova/transformers. Loads once, caches in IndexedDB, works without internet, supports 58 language pairs, hosted on GitHub Pages.
https://github.com/harisnae/multilingual-translator-offline

[3] Deploying ONNX Runtime Web | onnxruntime
https://onnxruntime.ai/docs/tutorials/web/deploy.html

[4] [5] [6] Translation Apps Market Report | Global Forecast From 2025 To 2033
https://dataintelo.com/report/global-translation-apps-market

[7] [19] [22] Offline speech recognition with Whisper: Browser + Node.js implementations
https://www.assemblyai.com/blog/offline-speech-recognition-whisper-browser-node-js

8 Whisper OpenAI In-Browser for Offline Audio Transcription | Data Science Collective
https://medium.com/data-science-collective/implementing-whisper-openai-in-browser-for-offline-audio-transcription-adab61be7af7

13 15 Web Speech API - Web APIs | MDN
https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

14 Offline Voice Recognition in a Web Browser - Picovoice
https://picovoice.ai/blog/offline-voice-ai-in-a-web-browser/

20 Designing for Web Accessibility – Tips for Getting Started - W3C
https://www.w3.org/WAI/tips/designing/