Rakuten Mobile Inc. Website Localization Project Requirements and Specifications

1. Introduction

- 1.1 Purpose

This document outlines the comprehensive requirements for the localization and translation of Rakuten Mobile Inc.'s (RMI) website. As RMI continues its expansion into international markets, including recent ventures into Europe with the launch of its network in Germany, the need for effective communication across multiple languages has become increasingly critical. The website, currently in Japanese, will be localized and translated into English, Simplified Chinese, Traditional Chinese, and Korean (South Korea) to support this global outreach.

This document details the procedures for localizing the source code within RMI's "src" directory, focusing on specific file types including JavaScript (JS), Embedded JavaScript (EJS), TypeScript (TS), TypeScript XML (TSX), Vue.js Single File Components (VUE), JavaScript Object Notation (JSON), and Comma-Separated Values (CSV). The goal is to programmatically identify, mark, and extract all translatable strings from these files, converting them into JSON resource files for subsequent translation.

Furthermore, this document addresses RMI's request to utilize Generative AI to generate human-readable JSON keys during this process. After thorough evaluation, however, it was determined that Generative AI may not be the most suitable approach for this specific task due to potential challenges in ensuring accuracy, consistency, and contextual relevance. Instead, this document proposes a more reliable, conventional method for generating these keys, aiming to meet RMI's localization needs while maintaining the highest standards of quality and efficiency.

- 1.2 Scope

The scope of this project is strictly confined to the localization and translation of specific file types within the source code of Rakuten Mobile Inc.'s (RMI) website, located in the "src" directory. The following key points outline the scope:

- Targeted Directories and File Types:

- The localization process will exclusively target the following file types within the "src" directory:

- Embedded JavaScript (EJS)

- JavaScript (JS)

- Vue.js Single File Components (VUE)

- TypeScript (TS)

- TypeScript XML (TSX)

- Comma-separated values (CSV)

- JavaScript Object Notation (JSON)

- The process will not include files located in the "dist" directory or any other directories outside of "src".

- String Extraction and Localization:

- All translatable strings from the targeted files within the "src" directory will be programmatically identified, marked, and extracted.

- The extracted strings will be populated into JSON resource files stored in the following directory structure:

- For general localization, JSON files will be stored in `public/locales/[locale]/` folders, where `[locale]` represents the language and region code (e.g., `en`, `zh-CN`, `ko`, etc.).

- Specific Folder Localization:

- For the `src/components/include/common` directory:

- All strings from the targeted file types in this directory will be consolidated into a single JSON file named `common.json`.

- This file will be stored at `public/locales/components/include/common/common.json`.

- For the `src/\_ejs/inc` directory:

- All strings from the targeted file types in this directory will be consolidated into a single JSON file, either named `common.json` or `inc.json`.

- This file will be stored at `public/locales/\_ejs/inc/common.json` (or `inc.json`).

- Exclusion of Direct HTML Localization:

- The project will not involve the direct localization of HTML files located in the "dist" directory.

- The localization focus will be solely on the EJS, JS, VUE, TS, TSX, CSV, and JSON files within the "src" directory.

- RMI will be responsible for running their build process on the localized "src" directory to generate the localized HTML files in the "dist" directory after the localization of the source files.

- Summary of Exclusions:

- No localization efforts will be directed at:

- HTML files within the "dist" directory.

- Any files or directories outside the specified "src" directory.

This focused scope ensures that the localization process aligns with RMI's requirements while maintaining the integrity of their build process, allowing for seamless integration of the localized content into the final website.

- 1.3 Definitions, Acronyms, and Abbreviations

This section provides definitions and explanations for the terms, acronyms, and abbreviations used throughout this document to ensure clarity and consistency.

- Localization:

- The process of adapting software or content to meet the language, culture, and other specific requirements of a target market or region. In this context, it refers to translating and adjusting the website’s content to suit the languages and cultural norms of different regions, including English, Simplified Chinese, Traditional Chinese, and Korean.

- Generative AI:

- A type of artificial intelligence that is capable of generating new content, such as text, images, or code, based on patterns it has learned from existing data. In this project, Generative AI was considered for generating human-readable JSON keys for translatable strings.

- JavaScript (JS):

- A dynamic programming language that is commonly used to create interactive effects within web browsers. JS files in the "src" directory are among the file types targeted for localization.

- Embedded JavaScript (EJS):

- A simple templating language that lets you generate HTML markup with plain JavaScript. EJS files in the "src" directory are targeted for localization to adapt content dynamically.

- TypeScript (TS):

- A strongly typed programming language that builds on JavaScript, providing optional static typing. TS files in the "src" directory are also included in the localization process.

- TypeScript XML (TSX):

- A syntax extension for TypeScript that allows embedding of XML-like syntax directly into TypeScript code. TSX files are particularly common in React applications and are included in the scope of localization.

- Vue.js Single File Components (VUE):

- A format used in Vue.js applications where HTML, JavaScript, and CSS are combined in a single file. VUE files in the "src" directory will be localized.

- JavaScript Object Notation (JSON):

- A lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON files are used in this project to store the translatable strings extracted from the source code.

- Comma-Separated Values (CSV):

- A simple file format used to store tabular data, such as spreadsheets or databases, in plain text. CSV files in the "src" directory are also subject to localization.

- \_ejs/inc Folder:

- A specific directory within the "src" directory where EJS files are stored. Strings extracted from files in this folder will be consolidated into a single JSON file (either `common.json` or `inc.json`).

- src/components/include/common Folder:

- Another specific directory within the "src" directory where various file types (e.g., EJS, JS, VUE, TS, TSX, CSV, JSON) are stored. Strings extracted from files in this folder will be consolidated into a single JSON file named `common.json`.

- Dist Folder:

- The distribution folder where the final compiled and minified code is stored, usually after a build process. The dist folder is not targeted for localization; instead, the localization is applied to the source files in the "src" directory, and the client will run their build process to generate the localized HTML files.

- Build Process:

- The sequence of automated steps that transforms source code into an executable program, or in the context of web development, into the final set of files (e.g., HTML, CSS, JS) that will be served to users. In this project, the client will handle the build process to generate localized HTML files from the localized source code.

- 1.4 References

This section lists the documents, resources, and references that are relevant to the source code localization process for Rakuten Mobile Inc.'s website project. These references provide additional context, guidelines, and standards that support the requirements and procedures outlined in this document.

* W3C Internationalization (i18n) Best Practices
  + W3C Internationalization Overview: <https://www.w3.org/International/i18n-drafts/getting-started/index.html>
  + This resource provides best practices for internationalization, ensuring that web content is easily adaptable for various languages and regions.
* Unicode Consortium: Unicode Standard
  + The Unicode Standard: <https://www.unicode.org/standard/standard.html>
  + The Unicode Standard specifies the representation of text in modern software products and standards, which is crucial for ensuring the correct handling of multilingual text during the localization process.
* JavaScript (ECMAScript) Language Specification
  + <https://tc39.es/ecma262/>
  + The official specification for JavaScript (ECMAScript), which is relevant for understanding the structure and syntax of JS and TS files involved in the localization process.
* Vue.js Documentation
  + Vue.js Guide: <https://vuejs.org/guide/introduction.html>  
     - Official documentation for Vue.js, a progressive JavaScript framework used in the development of interactive web interfaces, which includes guidelines for handling localization within Vue components.
* TypeScript Documentation
  + TypeScript Docs: <https://www.typescriptlang.org/docs/>
  + Comprehensive documentation on TypeScript, including details on its type system, syntax, and integration with JavaScript, relevant for localizing TS and TSX files.
* JSON Documentation
  + Introducing JSON: <https://www.json.org/json-en.html>
  + Documentation on JSON, the data format used to store and transfer the translatable strings extracted during the localization process. The client requested that original JSON files are duplicated for each locale.
* CSV Format Specification
  + RFC 4180 - Common Format and MIME Type for CSV Files: <https://tools.ietf.org/html/rfc4180>
  + The specification that defines the CSV file format, which is relevant for understanding how to handle and localize CSV files within the project. However, the client requested a duplication procedure for each locale for CSV files.

2. Project Description

- 2.1 Background

Rakuten Mobile Inc. (RMI) has initiated a project to localize and translate its website to cater to a global audience. Originally developed in Japanese, the website needs to be adapted to support multiple languages, specifically English, Simplified Chinese, Traditional Chinese, and Korean (South Korea). This localization effort is essential to ensure that RMI's users across various regions can interact with the website in their preferred languages, thereby improving accessibility and user experience.

The project focuses on localizing specific file types within the "src" directory of RMI's website source code. These file types include Embedded JavaScript (EJS), JavaScript (JS), Vue.js Single File Components (VUE), TypeScript (TS), TypeScript XML (TSX), Comma-Separated Values (CSV), and JavaScript Object Notation (JSON) files. These files contain the bulk of translatable content, such as user-facing text and other elements that need to be adapted to different languages. The project’s objective is to systematically identify, extract, and organize these translatable strings into JSON resource files, which will then be used for translation and later reintegration into the localized website.

As part of the project requirements, RMI expressed interest in utilizing Generative AI to automate the generation of human-readable JSON keys for the extracted strings. This approach was considered to potentially streamline the process, reducing manual effort and ensuring consistency in key generation. However, upon evaluation, several challenges were identified with using Generative AI for this task. These challenges include ensuring the contextual accuracy, relevance, and consistency of the AI-generated keys, which are critical for the success of the localization process.

Given these findings, the project will proceed with conventional methods for generating JSON keys. These methods are more reliable and better suited to meet the quality standards required by RMI. By adopting these proven practices, the project aims to deliver a high-quality localized website that meets RMI’s goals and provides an optimal experience for its global user base.

- 2.2 Goals and Objectives

The primary goals and objectives of the localization project for Rakuten Mobile Inc. (RMI) are focused on delivering a high-quality, accurate, and consistent localized version of RMI’s website that effectively meets the needs of a global audience. The specific goals and objectives are as follows:

* Accurate Localization:
  + Ensure that all translatable content within the targeted file types (EJS, JS, VUE, TS, TSX, CSV, JSON) is accurately identified, extracted, and translated. The project aims to maintain the original intent, meaning, and context of the content while adapting it to the cultural and linguistic norms of the target audiences.
* Consistency in Localized Content:
  + Achieve consistency in the localized content across all languages by using standardized processes for string extraction, translation, and JSON key generation. This includes the use of conventional methods for generating human-readable JSON keys, which are essential for maintaining uniformity across different parts of the website.
* Efficient String Extraction and Organization:
  + Implement a systematic approach to extract all translatable strings from the specified source code directories. These strings will be organized into JSON resource files, ensuring they are easily accessible for translation and reintegration into the website.
* Global Reach and Accessibility:
  + Expand RMI’s ability to reach and engage with a global audience by providing website content in multiple languages. This objective is aligned with RMI’s broader strategy of international expansion, particularly in regions where English, Simplified Chinese, Traditional Chinese, and Korean are predominantly spoken.
* Cultural Sensitivity and User Experience:
  + Enhance the user experience by ensuring that the localized content is culturally sensitive and tailored to the preferences of users in different regions. This includes not only the translation of text but also the adaptation of any cultural references, idioms, or visual elements that are part of the website content.
* Seamless Integration and Deployment:
  + Ensure that the localized content is seamlessly integrated back into the website’s source code. The project aims to allow RMI to run their build process on the localized source code, generating the final HTML files in the "dist" directory, ready for deployment without requiring additional localization efforts.
* Meeting Client Expectations:
  + Fulfill the specific requests and requirements set forth by RMI, including the consideration and evaluation of Generative AI for JSON key generation. The project will provide a detailed explanation of why conventional methods are recommended, ensuring that RMI’s expectations are met with transparency and professionalism.

These goals and objectives will guide the localization process, ensuring that the final product not only meets RMI’s requirements but also enhances the accessibility and usability of the website for a global audience.

- 2.3 Stakeholders

The success of the localization project depends on the coordinated efforts of various stakeholders, each of whom plays a critical role in different aspects of the project. The key stakeholders identified for this project are as follows:

- Rakuten Mobile Inc. (RMI) – The Client:

- RMI is the primary client for this project and the owner of the website undergoing localization. RMI's responsibilities include defining the project’s scope, providing requirements, and reviewing deliverables to ensure they meet their expectations. Key decision-makers from RMI will provide approval at various stages of the project, including the final acceptance of localized content.

- RMI's Product Management Team:

- This team within RMI is responsible for overseeing the overall product strategy and ensuring that the localized website aligns with RMI’s business goals. They are involved in setting the project’s objectives, timelines, and key performance indicators (KPIs).

- RMI's Marketing Team:

- The marketing team plays a crucial role in ensuring that the localized content resonates with the target audiences in different regions. They will collaborate closely with the development and translation teams to provide cultural insights and review content for marketing accuracy and appeal.

- Localization Development Team:

- This team is responsible for the technical execution of the localization process. Their tasks include identifying and extracting translatable strings from the source code, generating JSON resource files, and integrating the translated content back into the website. The development team will also ensure that the localized content functions correctly within the website’s framework.

- Translation and Linguistic Experts:

- The translators and linguistic experts are tasked with translating the extracted strings into the target languages (English, Simplified Chinese, Traditional Chinese, and Korean). They ensure that the translations are accurate, contextually appropriate, and culturally sensitive.

- Quality Assurance (QA) Team:

- The QA team is responsible for testing the localized content to ensure it meets quality standards. This includes validating the accuracy of translations, checking for consistency in JSON key usage, and ensuring the website functions as intended after localization. The QA team will conduct both linguistic and functional testing before the final deployment.

- Project Manager:

- The project manager oversees the entire localization process, ensuring that the project stays on schedule, within budget, and meets all defined objectives. The project manager acts as the primary point of contact between RMI and the various teams involved in the project, facilitating communication and resolving any issues that may arise.

- External Vendors or Partners:

- If any external vendors or partners are involved, such as translation agencies or additional software tools, they will also be considered stakeholders. Their role will be to provide specialized services or tools that support the localization process.

- End Users:

- While not directly involved in the project’s execution, the end users of RMI’s website—customers from different regions who will interact with the localized content—are the ultimate stakeholders. The success of the localization project will be measured by how well it meets the needs and expectations of these users.

Each of these stakeholders plays a vital role in the successful completion of the localization project. Effective communication and collaboration among all parties are essential to ensure that the project delivers a website that is not only fully localized but also aligned with RMI's global business objectives.

- 2.4 Assumptions and Dependencies

The successful execution of this localization project is based on several key assumptions and dependencies. These factors must be acknowledged and addressed to ensure the smooth progress and completion of the project. The following assumptions and dependencies have been identified:

Assumptions:

- Source Code Structure:

- It is assumed that the source code to be localized is contained within a single main directory named "mno\_marketing-master." This directory is the master branch of the source code repository hosted on Bitbucket. The primary focus will be on the "src" directory within this main directory, where the source files targeted for localization are located.

- Availability of the "src" Directory:

- The "src" directory, which houses the source code that needs to be localized, is assumed to be fully available and up-to-date. This directory contains various subfolders with different file types, including EJS, JS, VUE, TS, TSX, CSV, and JSON files. The assumption is that all necessary files for localization are contained within this "src" directory and that no additional files outside of this directory will require localization.

- Version Control:

- It is assumed that the "mno\_marketing-master" directory is the most current and stable version of the source code. Any updates or changes to the codebase that occur during the localization process will be communicated and integrated as needed. The version control system (Bitbucket) is assumed to be functioning properly, allowing for efficient tracking and management of changes.

- Access to Source Code:

- All relevant stakeholders, including the localization development team, are assumed to have the necessary access to the "mno\_marketing-master" repository on Bitbucket. This access is required for pulling the latest code, making necessary modifications, and pushing updates.

Dependencies:

- Localization Tools and Scripts:

- The project depends on the availability and functionality of specific tools and scripts used for identifying, extracting, and managing translatable strings. These tools must be compatible with the file types in the "src" directory and capable of generating JSON resource files as required by the project.

- JSON Resource Files:

- The creation and management of JSON resource files are dependent on the proper extraction of strings from the source code. The structure and content of these files are critical for the translation process and must be handled consistently across all targeted file types and directories.

- Translation Management System (TMS):

- The project assumes that the external Translation Management System (TMS) used for translating the extracted strings is fully operational and capable of integrating with the generated JSON files. The TMS must support the languages required for this project and be able to handle the volume of content to be localized.

- Testing and Validation Environments:

- The availability of testing and validation environments is a key dependency. These environments must mirror the production setup to accurately test the localized content before final deployment. The project relies on these environments to ensure that the localized website functions correctly and meets all quality standards.

- Client Feedback and Approval:

- The project’s progress and completion are dependent on timely feedback and approval from RMI at various stages. This includes reviews of extracted strings, translated content, and the final localized product. Any delays in feedback or approval may impact the overall timeline.

These assumptions and dependencies highlight the critical factors that must be in place for the project to proceed smoothly. Addressing these elements proactively will help mitigate risks and ensure the successful localization of RMI’s website.

3. Desired Localization Procedure (Client's Request)

- 3.1 File Types and Targeting

The client, Rakuten Mobile Inc. (RMI), has specified a clear and focused localization procedure that targets specific file types and directories within the website’s source code. The primary area of interest for localization is the "src" directory, which contains the core source code files that will be adapted for different languages. The specific file types and directories targeted for localization are as follows:

Targeted File Types:

- Embedded JavaScript (EJS):

- EJS files are utilized for dynamic content generation within the website. These files are key targets for localization as they often contain strings that are rendered into the final HTML content.

- JavaScript (JS):

- JavaScript files, responsible for the interactive and functional aspects of the website, include translatable strings that may appear in user interfaces, alerts, and other dynamic elements. These files are essential for ensuring that all user-facing content is accurately translated.

- Vue.js Single File Components (VUE):

- Vue.js components are used for building the website’s user interface. These VUE files combine HTML, JS, and CSS, making them significant for localization due to the embedded translatable strings within the template sections.

- TypeScript (TS):

- TypeScript files, which add static types to JavaScript, are also targeted for localization. These files may contain user-facing text that requires translation to ensure consistency across the application.

- TypeScript XML (TSX):

- TSX files, commonly used in frameworks like React, combine TypeScript and JSX syntax. They are included in the localization process because they often embed translatable strings within the code, particularly in UI components.

- Comma-Separated Values (CSV):

- CSV files, which store data in a tabular format, may contain strings that are displayed on the website or used in data-driven content. These files are targeted to ensure that all relevant data is translated appropriately.

- JavaScript Object Notation (JSON):

- JSON files are used to store configuration data, content, or settings that may include translatable strings. Localizing these files is critical for ensuring that all dynamic content adheres to the target languages.

Targeted Directories within the "src" Folder:

- All Directories in "src":

- The entire "src" directory is targeted for localization. This includes all subdirectories containing the various file types (EJS, JS, VUE, TS, TSX, CSV, JSON) that hold translatable strings. The goal is to ensure that every part of the website's source code is localized, encompassing all directories within "src" without exception.

- src/components/include/common:

- Within the "src" directory, the `components/include/common` subdirectory is specifically highlighted for containing common components and files that are reused across multiple parts of the website. The client has requested that all translatable strings within this directory be consolidated into a single JSON file named `common.json`, which will be stored in `public/locales/components/include/common/common.json`.

- src/\_ejs/inc:

- Similarly, the `src/\_ejs/inc` directory is noted for containing EJS files that are included in various sections of the website. The client has requested that all translatable strings within this directory be consolidated into a single JSON file, either named `common.json` or `inc.json`, to be stored in `public/locales/\_ejs/inc/common.json` (or `inc.json`).

These highlighted directories within the "src" folder are critical because they contain common strings used throughout the website, making it efficient to group them into single JSON files for easier management and translation. However, the entire "src" directory, including all other subdirectories, is also within the scope of this localization project to ensure comprehensive coverage of all translatable content.

Exclusions:

- Dist Directory:

- The "dist" directory, which contains the compiled output files such as HTML, is explicitly excluded from the localization process. The focus is solely on the source code in the "src" directory, and the client will handle the build process to generate localized HTML files after the localization of the source files.

By targeting these specific file types and directories, the localization process is streamlined to focus on the most impactful areas of the website’s source code, ensuring that all relevant content is appropriately localized while adhering to the client’s requirements.

- 3.2 Translatable Strings Identification

Rakuten Mobile Inc. (RMI) has provided specific guidelines and expectations for identifying translatable strings within the source files as part of the localization process. The following outlines the client’s expectations regarding the format and methodology for marking and extracting these translatable strings across the various file types targeted for localization.

* Identification Format and Extraction Methods:
  + TypeScript XML (TSX):

Translatable strings within TSX files should be identified using the format `{ t('namespace.key') }`. This format ensures that each string is associated with a unique namespace and key, facilitating organized and efficient translation. The `namespace.key` structure helps maintain context and consistency across the translations.

* + TypeScript (TS):

For translatable strings in TypeScript files (excluding those in TSX), the identification format is `t('namespace.key')`. This method is consistent with the TSX approach, ensuring that all strings extracted from TypeScript files are uniformly structured. The extraction process should focus on isolating strings intended for translation while maintaining their connection to the associated namespace.

* + Embedded JavaScript (EJS):

In EJS files, translatable strings should be marked using the format `${{ namespace.key }}$`. This format encapsulates the translatable content in a way that aligns with EJS’s syntax, allowing for seamless integration into the template system while maintaining clear identification for extraction.

* + Vue.js Single File Components (VUE):

For VUE files, the format for identifying translatable strings is `{{ $t('namespace.key') }}`. This format leverages Vue.js’s built-in localization capabilities, ensuring that each string is properly marked for extraction and subsequent translation within the component-based architecture of Vue.js.

* + JavaScript (JS):

In JavaScript files, translatable strings should be identified using two methods depending on the context:

For interpolation: `${translate('key', langs)}` is used to mark strings within template literals that involve dynamic content.

For static values: `translate('key', langs)` should be used to mark translatable strings that do not involve interpolation.

RMI has indicated that some modifications to this format may be applied later, based on the specific needs of the project as it progresses.

* + JavaScript Object Notation (JSON):

For JSON files, the client expects that the strings will be duplicated for each locale. This approach involves creating separate JSON files for each target language, ensuring that all translatable content is properly organized and accessible for translation.

* + Comma-Separated Values (CSV):

Similar to JSON, CSV files should also be duplicated for each locale. This method ensures that all translatable data within CSV files is available in the respective languages, making the localization process straightforward and consistent across different file formats.

* Client Expectations:

RMI expects that the translatable strings will be systematically identified and extracted from the source code using the formats outlined above. This structured approach is intended to ensure consistency across all file types and facilitate the efficient generation of JSON resource files that will be used for translation. Additionally, the client expects that any necessary modifications to the identification formats, particularly for JavaScript files, will be communicated and implemented as needed to meet the project’s requirements.

By adhering to these guidelines, the project aims to achieve a high level of accuracy and consistency in the localization process, ensuring that all user-facing content is properly translated and integrated into the localized website.

- 3.3 Use of Generative AI for JSON Key Generation

RMI expressed interest in utilizing Generative AI, specifically models like GPT-4, to automate the generation of human-readable JSON keys during the localization process. The intent behind this request was to streamline the key generation process, reduce manual effort, and ensure consistency across the various files being localized.

Client’s Request:

RMI proposed the use of Generative AI to generate JSON keys for the translatable strings identified within the targeted file types, including JavaScript (JS), Embedded JavaScript (EJS), TypeScript (TS), TypeScript XML (TSX), and Vue.js Single File Components (VUE). The expectation was that AI could automatically create meaningful and contextually appropriate JSON keys that would be human-readable, thereby facilitating the organization and management of these keys in the localization process.

- 3.4 Desired Output Format

3.4 Desired Output Format

The client has specified the desired format and structure for the output files resulting from the localization process. These files are crucial for ensuring that the localized content is correctly integrated into the website, enabling seamless translation across multiple languages. The primary output files include JSON files with generated keys (whether AI-generated or conventionally generated), as well as other formats like CSV, which are duplicated for each locale.

JSON Files:

- Structure and Naming Conventions:

- Each JSON file should be structured with clear and organized key-value pairs. The keys should be human-readable, meaningful, and contextually relevant to the content they represent. The values associated with these keys will be the translatable strings in the respective languages.

- The JSON files should adhere to a consistent naming convention based on the directory and file type they are derived from. For instance, files generated from the `src/components/include/common` directory might be named `common.json`, while those from `src/\_ejs/inc` could be named `inc.json` or `common.json`, depending on the specific instructions.

- File Path as Namespace:

- The JSON keys should use the file path as the namespace to maintain organization and context. For example, a string in the `src/components/header.js` file might have a key like `src.components.header.title`, where `src.components.header` serves as the namespace derived from the file path, and `title` is the specific key for the translatable string.

- In cases where AI is utilized for key generation, the keys should still adhere to this structure, with additional validation to ensure that the AI-generated keys are logical and consistent with the overall naming conventions.

- Locale-Specific JSON Files:

- Each translatable string should be duplicated across multiple JSON files, one for each locale. For instance, the same `common.json` file will exist in multiple directories corresponding to each target language, such as:

- `public/locales/en-US/common.json` for English (United States)

- `public/locales/zh-CN/common.json` for Simplified Chinese

- `public/locales/zh-TW/common.json` for Traditional Chinese

- `public/locales/ko-KR/common.json` for Korean (South Korea)

- Content Duplication and Consistency:

- The content of the JSON files should be duplicated and adapted for each locale while maintaining consistency in the key structures. This approach ensures that the translation process can proceed smoothly, with translators working from a consistent set of keys across all languages.

CSV Files:

- Duplicated for Each Locale:

- Similar to JSON files, CSV files containing translatable strings should be duplicated for each target locale. The CSV format will allow for easy handling of tabular data that needs to be localized.

- The CSV files should be named according to the locale and the content they represent, ensuring clarity and organization. For instance, a CSV file for English (United States) might be named `data\_en-US.csv`, while its Simplified Chinese counterpart would be `data\_zh-CN.csv`.

File Storage and Organization:

- Directory Structure:

- All output files, including JSON and CSV files, should be stored in a clearly defined directory structure under `public/locales/`. Each locale will have its own subdirectory where the localized files are stored. For example:

- `public/locales/en-US/` for English (United States) files

- `public/locales/zh-CN/` for Simplified Chinese files

- `public/locales/zh-TW/` for Traditional Chinese files

- `public/locales/ko-KR/` for Korean (South Korea) files

- Common Files Handling:

- Files that contain common strings used across multiple parts of the website, such as those in `src/components/include/common` and `src/\_ejs/inc`, should be handled with particular attention to ensure that their keys are consistent and meaningful across all locales.

The client’s desired output format focuses on ensuring that all localized content is well-structured, organized, and easily manageable across different languages. By adhering to these guidelines, the project will produce output files that facilitate an efficient and effective translation process, ultimately leading to a high-quality localized website.

- 3.5 Additional Client Requirements

At this stage, the client has not specified any additional requirements beyond those already detailed in the previous sections of this document. The project is therefore limited to the current scope, which includes the identification, extraction, and localization of translatable strings within the specified file types and directories, and the organization of output files in the desired format.

Should any additional requirements arise during the project, they will be documented and addressed accordingly to ensure that all aspects of the localization process meet the client’s expectations and objectives.

4. Proposed Localization Procedure (Engineer’s Approach)

- 4.1 File Procedure

The engineer proposes to use Python, along with its robust set of libraries, to develop a custom solution for scanning and identifying relevant files and translatable strings within the "src" directory. This approach leverages Python’s flexibility and the availability of powerful libraries for file handling, text processing, and pattern matching.

Methodology:

1. Directory Scanning:

- The process begins with a comprehensive scan of the entire "src" directory. Python’s `os` and `glob` libraries will be utilized to recursively navigate through the directory structure, identifying all files that match the targeted file types: EJS, JS, VUE, TS, TSX, CSV, and JSON.

- The script will filter out any non-targeted files, ensuring that only the relevant files are processed for localization.

2. File Type Identification:

- Once the files are identified, the script will categorize them based on their file extensions (e.g., `.ejs`, `.js`, `.vue`, `.ts`, `.tsx`, `.csv`, `.json`). This categorization is crucial for applying the correct translatable string identification methods tailored to each file type.

3. String Identification and Extraction:

- The script will employ regular expressions (regex) to search for patterns corresponding to translatable strings in each file type. These patterns will be aligned with the formats specified by the client, such as `{ t('namespace.key') }` for TSX files, `t('namespace.key')` for TS files, `${{ namespace.key }}$` for EJS files, `{{ $t('namespace.key') }}` for VUE files, and `translate('key', langs)` for JS files.

- Python’s `re` library will be used to implement these regex patterns, allowing for efficient and accurate identification of translatable strings within the codebase.

4. Namespace Assignment:

- As each translatable string is identified, the script will automatically assign a namespace to it based on the file path. This namespace assignment ensures that all strings are organized in a consistent and logical manner, corresponding to their location within the directory structure.

5. Error Handling and Logging:

- The script will include error handling mechanisms to manage any issues that arise during the scanning and identification process, such as inaccessible files or unrecognized patterns.

- A logging system will be implemented to record the progress of the scan, including details on which files were processed, how many strings were identified, and any errors encountered. This log will be crucial for troubleshooting and ensuring that the process runs smoothly.

Implementation Details:

- Libraries and Tools:

- The proposed solution will be implemented using Python’s standard libraries, such as `os`, `glob`, and `re`, along with any additional libraries required for handling specific tasks (e.g., `pandas` for CSV manipulation if needed).

- The script will be modular, allowing for easy updates or adjustments as needed based on the evolving requirements of the localization project.

- Performance Considerations:

- The script will be optimized for performance to handle large volumes of files efficiently. Techniques such as file buffering and asynchronous processing may be employed to minimize processing time, especially for larger files or directories.

By utilizing Python and its extensive library ecosystem, the engineer’s approach to file scanning and identification is designed to be both powerful and flexible, ensuring that all relevant translatable strings are accurately identified and prepared for the localization process.

- 4.2 Evaluation of Generative AI for JSON Key Generation

Preliminary Evaluation and Observations:

During the preliminary evaluation of using GPT-4 for this task, several challenges were identified:

- Token Limitations:

- Most files tested in each category (JS, EJS, TS, TSX, and VUE) encountered the `max\_tokens` limit error. The GPT-4 model has a maximum token limit of 8192 tokens, which includes both input and output tokens. Many files exceeded this limit, leading to incomplete or failed processes.

- Switching to Higher Capacity Models:

- In an attempt to resolve this issue, testing was conducted using the GPT-4-32k model, which has a larger token limit of 32,768 tokens. However, some files still exceeded this limit, particularly those with an average of 33,000 tokens. This issue persisted, indicating that the problem was not fully resolved by simply switching models.

- Chunking Attempts:

- To work around the token limitations, chunking of the files' contents was attempted. However, this approach also proved problematic. The chunking process often led to the same `max\_tokens` limit error, and the model’s varying approach to handling chunks introduced inconsistencies. Moreover, chunking raised concerns about potentially breaking the original code structure, as the model had limited control over maintaining the correct scope and context when processing in smaller chunks.

- Inconsistent Results:

- Even when files passed the token limit without errors, the results were not always satisfactory in terms of form and correctness. This inconsistency poses a significant risk to the reliability and quality of the localization process.

Next Steps and Considerations:

Given these challenges, there is consideration of exploring LangChain, a tool designed to handle chunking more smoothly while maintaining consistency and scope. However, there are still concerns about the ability to ensure consistent and contextually accurate results with this approach.

The preliminary findings suggest that while Generative AI has potential in automating parts of the localization process, it may not be the most suitable approach for generating JSON keys in this context, particularly given the token limitations and inconsistent results observed during testing. The complexity of the files and the need for high accuracy and reliability in the localization process may necessitate a return to more conventional methods for JSON key generation.

RMI’s expectations for using AI in this capacity have been carefully considered, and the evaluation highlights the need for a balanced approach that considers both the potential benefits and the limitations of AI in the localization workflow.

- 4.3 Comparison of Conventional Approach with AI Approach

When considering the methodologies for generating JSON keys during the localization process, both conventional methods and Generative AI offer distinct advantages and challenges. This section compares the two approaches, highlighting the key reasons for recommending conventional methods over AI in this project.

1. Accuracy and Consistency:

- Conventional Approach:

- The conventional method involves manually or programmatically generating JSON keys based on predefined structures, such as using the file path as a namespace. This approach ensures high accuracy and consistency, as the key generation is directly tied to the logical structure of the source code. By maintaining a clear and consistent naming convention, the risk of errors or inconsistencies in key naming is minimized.

- AI Approach:

- Generative AI, such as GPT-4, has the capability to generate human-readable JSON keys based on context and content. However, during preliminary testing, it was observed that AI-generated keys were inconsistent and sometimes contextually inappropriate. The AI struggled with maintaining a consistent naming convention across different files, leading to potential confusion and errors in the localization process.

Advantage: The conventional approach provides greater control and reliability in generating consistent and contextually appropriate JSON keys.

2. Handling Large Files and Token Limitations:

- Conventional Approach:

- The conventional method does not face limitations related to file size or token count, as it directly interacts with the source code through predefined scripts and logic. This approach can handle large files efficiently without concerns about exceeding any processing limits.

- AI Approach:

- During testing, the AI approach encountered significant challenges with token limitations. The GPT-4 model, even with the expanded `gpt-4-32k` version, struggled to process files exceeding the token limit, resulting in incomplete outputs and errors. While chunking was attempted as a workaround, it led to additional issues, such as breaking code structure and further inconsistencies.

Advantage: The conventional approach is more reliable for handling large files, ensuring that the entire content is processed without encountering token-related errors.

3. Control Over Output:

- Conventional Approach:

- The conventional method offers full control over the output, allowing the engineer to fine-tune the process, apply custom logic, and ensure that the generated keys align perfectly with the project’s requirements. This control is crucial for maintaining the integrity of the codebase and ensuring that the localized content is correctly structured.

- AI Approach:

- The AI approach, while flexible, lacks the precision and control offered by conventional methods. The AI’s output is based on probabilistic models, which can lead to unpredictable results, especially when dealing with complex or nuanced content. The lack of control over AI-generated keys can result in outputs that require significant post-processing, negating the time-saving benefits of automation.

Advantage: The conventional approach offers precise control over the output, ensuring that the generated JSON keys are exactly as needed without requiring extensive post-processing.

4. Scalability and Maintenance:

- Conventional Approach:

- The conventional method, once set up, is easily scalable and maintainable. Scripts can be reused, modified, and extended as needed, making the approach highly adaptable to changes in project requirements. Additionally, the process is transparent, with clear logic that can be understood and adjusted by future developers or maintainers.

- AI Approach:

- While AI can potentially scale by processing large amounts of data, the scalability is hindered by token limits and the need for constant fine-tuning to achieve the desired results. Furthermore, AI models may need to be retrained or reconfigured for different projects, adding to the maintenance overhead.

Advantage: The conventional approach is more scalable and easier to maintain over time, providing long-term benefits for the project’s sustainability.

5. Implementation Complexity:

- Conventional Approach:

- The conventional method involves setting up scripts and logic that are straightforward to implement with the appropriate programming knowledge. Once established, the process runs with minimal intervention, making it relatively simple to manage.

- AI Approach:

- Implementing AI for JSON key generation introduces additional complexity, including managing token limits, configuring models, and handling unexpected output. The complexity of setting up and maintaining AI-driven processes can outweigh the potential benefits, especially when the results are inconsistent.

Advantage: The conventional approach is simpler to implement and manage, reducing the risk of errors and ensuring a smoother workflow.

Conclusion:

While Generative AI presents an intriguing option for automating JSON key generation, the conventional approach is recommended for this project due to its superior accuracy, consistency, control, and scalability. The challenges encountered during AI testing, particularly related to token limitations and inconsistent outputs, highlight the risks involved with relying on AI for this task. The conventional methods, by contrast, offer a proven and reliable solution that aligns more closely with the project’s requirements and ensures the highest quality in the localization process.

- 4.4 Handling Edge Cases

In the localization process, certain edge cases such as dynamic content and embedded variables within strings present unique challenges. The engineer acknowledges these complexities and will carefully handle such edge cases where applicable to ensure the integrity and functionality of the localized content.

1. Dynamic Content Handling:

Dynamic content, which is generated or altered at runtime, can be difficult to identify and extract for localization. While the general process for string extraction will be automated, the engineer will carefully review and address dynamic content where necessary. This careful approach ensures that critical elements are not overlooked and that the dynamic behavior of the content is preserved during localization.

2. Embedded Variables within Strings:

Strings that contain embedded variables, such as `${user.name}`, require careful handling to ensure that these variables remain functional and are not altered during translation. The engineer will carefully review strings with embedded variables and make necessary adjustments to ensure that only the translatable text is extracted while preserving the integrity of the embedded variables.

3. Manual Review of Edge Cases:

The engineer will conduct a manual review of all identified edge cases, focusing on those involving dynamic content and embedded variables. This review will involve checking the extracted strings to ensure that they are correctly formatted and that any necessary adjustments are made before proceeding with translation. By manually handling these edge cases, the engineer aims to maintain the quality and functionality of the localized content without overcomplicating the process.

Conclusion:

In handling edge cases, the engineer will prioritize simplicity and direct intervention, ensuring that complex scenarios are managed effectively without extensive automation or testing procedures. This approach allows for greater flexibility and ensures that all unique cases are addressed appropriately within the localization process.

5. Technical Specifications

- 5.1 Tools and Techniques

To effectively implement the localization procedure, the engineer will utilize a variety of tools and techniques designed to streamline the process of identifying, extracting, and organizing translatable content from the source code. The following outlines the key tools and techniques that will be employed in this project:

1. Python Programming Language:

- Python:

- Python will be the primary programming language used for developing scripts to automate the localization tasks. Its versatility and extensive libraries make it well-suited for handling file manipulation, string processing, and data extraction tasks efficiently.

2. Text Parsing and String Extraction Tools:

- Regular Expressions (Regex):

- Regex will be employed to define specific patterns for identifying translatable strings within the source files. Regex allows for precise string matching and extraction, tailored to the syntax of the various file types (e.g., JS, EJS, VUE, TS, TSX).

- Custom Python Scripts:

- The engineer will write custom Python scripts to automate the scanning, marking, and extraction of translatable strings. These scripts will use the `re` library for regex operations, the `os` and `glob` libraries for file navigation, and other Python tools to manage the extraction process efficiently.

3. File and Data Handling Libraries:

- OS and Glob Libraries:

- These libraries will be used for navigating the directory structure and identifying files that match the targeted extensions. They allow for recursive scanning of directories, ensuring that all relevant files within the "src" directory are processed.

- JSON Library:

- The Python `json` library will be used to create and manage JSON resource files. This includes writing the extracted strings to the appropriate JSON files, ensuring they are well-structured and ready for translation.

- CSV Library:

- For handling CSV files, the `csv` library in Python will be utilized. It enables the reading, writing, and duplication of CSV content across different locales, ensuring that all tabular data is appropriately localized.

4. Code Analysis Tools:

- Pylint:

- Pylint will be used for code analysis to ensure that the Python scripts adhere to coding standards and best practices. This tool will help identify potential issues in the scripts, such as syntax errors or inefficiencies, before they are deployed.

- Manual Review Techniques:

- While automated tools will handle most of the localization process, manual review techniques will be employed for edge cases or complex strings that require human oversight. This ensures that the final output maintains the intended functionality and quality.

5. Version Control and Collaboration:

- Git and Bitbucket:

- Version control will be managed using Git, with the codebase hosted on Bitbucket. This ensures that all changes to the scripts and localized content are tracked, allowing for collaboration, rollback, and auditability throughout the project lifecycle.

Conclusion:

By leveraging these tools and techniques, the engineer will be equipped to efficiently execute the localization process, ensuring that all translatable content is accurately identified, extracted, and prepared for translation. These tools provide a robust framework for managing the complexities of the localization project, while also allowing for flexibility and scalability as the project evolves.

6. Validation and Testing

- 6.1 Validation Process

The validation process for ensuring that all translatable strings are correctly identified and extracted will rely primarily on user testing and manual review. Given the scope of the project and the client's requirements, the following steps will be undertaken to validate the effectiveness of the localization process:

1. User Testing of the Source Code:

- Initial Deployment:

- After the translatable strings have been identified, extracted, and organized into JSON files, the localized version of the source code will be deployed for user testing. This initial deployment will allow the client and their team to interact with the localized content in a live environment.

- User Feedback:

- The client will conduct user testing to assess the accuracy and completeness of the localized strings. This testing will involve navigating the website, interacting with various elements, and ensuring that all user-facing text appears as expected in the target languages. The client may identify issues such as untranslated strings, incorrect translations, or misplaced text.

2. Manual Review:

- Client Review:

- The client will manually review the localized content as part of their validation process. This review will focus on checking that the extracted strings have been correctly identified and that they align with the intended meaning and context within the website's interface.

3. Issue Reporting and Resolution:

- Feedback Loop:

- Any issues identified during user testing or manual review will be reported back to the engineer. This feedback loop is critical for addressing any oversights or errors that may have occurred during the initial extraction and localization process.

- Issue Resolution:

- Upon receiving feedback, the engineer will investigate the reported issues, make necessary corrections, and redeploy the updated source code. This process ensures that all identified issues are resolved to the client's satisfaction.

4. Final Validation:

- Client Approval:

- Once all reported issues have been addressed, the client will conduct a final round of testing to validate the changes. If the client is satisfied with the results, they will approve the localized content for final deployment.

Conclusion:

The validation process relies heavily on user testing and client feedback to ensure that the localization meets the desired standards. By engaging the client directly in the validation process, the project ensures that any issues are promptly identified and resolved, leading to a high-quality final product that aligns with the client's requirements.

7. Deliverables

- 7.1 List of Deliverables

The following is a comprehensive list of deliverables for the localization project. These deliverables encompass all the key outputs necessary to ensure a successful and high-quality localization of the client’s website source code.

1. Localized Source Code Files:

- Fully Localized Source Code:

- The complete set of source code files that have been fully localized by marking all translatable strings. These files will include the necessary annotations or modifications to ensure that every translatable string is correctly identified and prepared for extraction into the appropriate JSON files.

2. Extracted JSON Files:

- Locale-Specific JSON Files:

- JSON files containing the translatable strings for each target language. Each locale will have its own set of JSON files organized according to the directory structure outlined in the project requirements.

- Namespace-Organized JSON Files:

- JSON files with keys organized by file paths serving as namespaces. This structure ensures that the strings are easily traceable back to their original location within the source code, facilitating future maintenance and updates.

- Duplicated JSON Files:

- JSON files will be duplicated for each target locale, ensuring that each language has its own corresponding JSON resource files. This duplication allows for organized and efficient translation management.

3. CSV Files:

- Locale-Specific CSV Files:

- CSV files for each target language, containing any translatable data that was identified within CSV formats. These files will be duplicated and adjusted for each locale.

4. Source Code Update Guidelines:

- Documentation:

- A comprehensive document detailing the guidelines for maintaining the localized source code. This documentation will cover best practices for updating the source code post-localization, ensuring that any future changes are consistent with the current localization framework.

- The documentation will include:

- Instructions on how to add new translatable strings and update existing ones.

- Guidelines for modifying or extending the JSON files as the website evolves.

5. Validation Feedback and Updates:

- Summary of User Testing and Issue Resolution:

- A summary document that captures the feedback received during the user testing phase and the corresponding updates made to the localized content. This summary will serve as a record of the validation process and ensure transparency regarding any changes implemented based on client feedback.

6. Final Report:

- Project Completion Report:

- A final report summarizing the entire localization project, including the processes used, challenges encountered, and how they were addressed. This report will also highlight any key insights or recommendations for the client regarding future localization efforts.

Conclusion:

These deliverables are designed to provide the client with all the necessary tools, documentation, and resources to maintain and expand the localized version of their website. By delivering a comprehensive set of outputs, the project ensures that the client has everything needed for both immediate deployment and long-term sustainability of the localized content.

- 7.2 Acceptance Criteria

The acceptance criteria for each deliverable are defined to ensure that all outputs meet both the client’s requirements and the engineer’s standards for quality and accuracy. These criteria will serve as the basis for evaluating the completeness and correctness of each deliverable before final approval.

1. Localized Source Code Files:

- Criteria:

- All translatable strings within the source code files must be accurately identified and marked according to the agreed-upon format.

- The source code must maintain its original functionality and integrity after localization, with no introduced errors or issues.

- The localized source code files should be fully compatible with the client’s build and deployment processes, ensuring that they integrate seamlessly into the existing development workflow.

2. Extracted JSON Files:

- Locale-Specific JSON Files:

- Each JSON file must contain all translatable strings accurately extracted from the source code, organized according to the correct locale.

- The JSON files should be free of syntax errors and formatted consistently to ensure easy readability and maintainability.

- Each JSON file must align with the namespace and key structure defined in the project requirements, ensuring traceability back to the original source code.

- Duplicated JSON Files:

- The duplicated JSON files for each locale must mirror the content and structure of the original JSON files, with all strings correctly duplicated and ready for translation.

- The duplication process must maintain consistency across all locales, ensuring that no strings are missing or incorrectly assigned.

3. CSV Files:

- Criteria:

- CSV files must include all relevant translatable data accurately identified and extracted from the source code.

- The files should be correctly duplicated for each target locale, with the content adjusted as needed for language-specific formatting or structure.

- Each CSV file must be error-free, with no missing data or formatting issues that could affect the translation or integration process.

4. Source Code Update Guidelines:

- Criteria:

- The documentation must clearly and comprehensively outline the guidelines for updating the localized source code, ensuring that future changes are consistent with the established localization framework.

- The guidelines should be easy to understand and implement, providing clear instructions for adding new translatable strings, updating existing ones, and modifying JSON files as necessary.

- The document should be reviewed and approved by both the engineer and the client to ensure it meets their standards and addresses all relevant aspects of source code maintenance post-localization.

5. Validation Feedback and Updates:

- Criteria:

- The summary document must accurately reflect the feedback received during user testing, including all reported issues and the corresponding updates made to the localized content.

- The updates implemented based on the feedback must be thoroughly tested and validated, ensuring that all issues are resolved to the client’s satisfaction.

- The document should provide a clear record of the validation process, including any decisions made regarding specific feedback or changes.

6. Final Report:

- Criteria:

- The final report must provide a comprehensive overview of the localization project, covering all key aspects of the process, challenges encountered, and how they were addressed.

- The report should include recommendations for future localization efforts, based on the insights gained during the project.

- The final report should be reviewed and approved by the client as a complete and accurate reflection of the project’s outcomes.

Conclusion:

Meeting these acceptance criteria is essential for ensuring that all deliverables not only fulfill the client’s requirements but also uphold the engineer’s standards for quality and reliability. By adhering to these criteria, the project will deliver high-quality outputs that are ready for deployment and future maintenance.

8. Project Management

- 8.1 Project Schedule

The project schedule outlines the timeline for the localization project, beginning with the project kickoff in mid-August 2024 and concluding by the end of September 2024. The schedule includes key milestones for different stages of the localization process, ensuring that the project progresses smoothly and meets the client’s requirements within the specified timeframe.

Week 1 (August 19 - August 31, 2024):

- Project Kickoff and Initial Setup:

- Finalize project requirements with the client.

- Set up the development environment and version control system.

- Begin scanning the "src" directory to identify all relevant files for localization.

Week 2 (September 1 - September 7, 2024):

- String Identification and Marking:

- Implement scripts for identifying and marking translatable strings in the targeted files.

- Begin the process of marking translatable strings in the localized source code files.

- Extraction of Translatable Strings:

- Extract the identified translatable strings and organize them into JSON files for each locale.

- Duplicate and structure the JSON and CSV files for the respective locales.

Week 3 (September 8 - September 14, 2024):

- Source Code Review and Adjustments:

- Conduct a review of the marked source code and extracted files to ensure accuracy and completeness.

- Make any necessary adjustments to the source code or JSON files based on the review.

- Initial Deployment for User Testing:

- Deploy the localized source code and resource files for initial user testing by the client.

- Begin collecting feedback from the client regarding the accuracy and functionality of the localized content.

Week 4 (September 15 - September 21, 2024):

- Feedback Integration and Issue Resolution:

- Address any issues or feedback provided by the client from the initial user testing.

- Make necessary updates to the localized content and redeploy for further testing if needed.

- Final Validation and Documentation:

- Perform final validation of the localized source code and ensure all deliverables meet the acceptance criteria.

- Prepare documentation, including source code update guidelines and a summary of the validation feedback.

Week 5 (September 22 - September 30, 2024):

- Final Client Review and Approval:

- Present the finalized localized content and documentation to the client for final review.

- Obtain client approval on all deliverables and make any last-minute adjustments if required.

- Project Closure:

- Deliver the final report summarizing the project and its outcomes.

- Complete any remaining administrative tasks and formally close the project.

Conclusion:

This timeline ensures that the localization project starts in mid-August 2024 and is completed within September 2024, with all key milestones met and deliverables provided to the client by the end of the month. By squeezing some tasks into the same weeks, the project remains on track to meet the client’s expectations within the shortened timeframe.

- 8.2 Resource Allocation

The resource allocation for the localization project includes a detailed overview of the personnel, tools, and time required to successfully complete the project within the specified timeline.

1. Personnel:

- Project Manager:

- The Project Manager will oversee the entire localization process, ensuring that milestones are met on time and that communication between the client and the engineering team is smooth and effective. The Project Manager will also be responsible for handling any issues that arise and ensuring that all deliverables meet the client's expectations.

- Software Engineer:

- The Software Engineer will be responsible for implementing the localization procedure, including writing and deploying the necessary scripts, marking and extracting translatable strings, and preparing the localized source code and JSON/CSV files. The engineer will also handle the manual review of edge cases and integrate feedback from user testing.

- Quality Assurance (QA) Specialist:

- The QA Specialist will assist with validating the localized content, conducting user testing, and ensuring that the final product is free from errors or issues. The QA Specialist will work closely with the client to gather feedback and will help verify that all corrections are made before the final deployment.

- Translator(s):

- Depending on the number of target languages, one or more translators will be required to translate the extracted strings into the desired languages (en-US, zh-CN, zh-TW, ko-KR). The translators will work with the localized JSON and CSV files to ensure that the translations are accurate and contextually appropriate.

2. Tools:

- Development Environment:

- A reliable development environment with access to the version control system (e.g., Git, Bitbucket) will be required for all team members. This environment will facilitate the development, testing, and deployment of the localized content.

- Python Programming Language:

- Python will be the primary tool used for automating the localization process. The engineer will utilize Python libraries such as `os`, `glob`, `re`, `json`, and `csv` to handle file operations, string extraction, and data management.

- Regular Expressions (Regex):

- Regex will be used extensively within Python scripts to identify and extract translatable strings from the source code.

- Translation Management System (TMS):

- A TMS may be used to manage the translation process, ensuring that the translators can work efficiently and that the localized content is consistently organized.

- Text Editors/Integrated Development Environments (IDEs):

- Text editors or IDEs such as Visual Studio Code, PyCharm, or Sublime Text will be used by the engineering team for writing and editing the source code and scripts.

- QA Tools:

- Tools such as browser-based testing environments and manual testing frameworks will be used by the QA Specialist to validate the localized content and ensure it meets the required standards.

3. Time Allocation:

- Project Kickoff and Initial Setup (Week 1 - Mid August):

- Approximately 10% of the total project time will be allocated to setting up the development environment, finalizing requirements, and beginning the initial file scanning process.

- Localization Implementation (Weeks 2 - 3):

- Around 30% of the project time will be allocated to the string identification, marking, extraction, and file preparation processes. This stage includes setting up the scripts, processing the source code, and organizing the extracted strings.

- Review and Testing (Weeks 3 - 4):

- Approximately 30% of the project time will be dedicated to reviewing the localized source code, deploying it for user testing, and integrating feedback from the client. This period will also involve making necessary adjustments and validating the updates.

- Final Validation and Documentation (Weeks 4 - 5):

- The remaining 30% of the project time will be focused on final validation, documentation preparation, final client review, and project closure.

Conclusion:

This resource allocation plan ensures that the right personnel, tools, and time are dedicated to each stage of the localization project. By carefully managing these resources, the project is set up to meet its goals efficiently and deliver high-quality localized content to the client by the end of September 2024.

- 8.3 Risk Management

Effective risk management is crucial to ensuring the success of the localization project. This section identifies potential risks that could impact the project, particularly focusing on technical challenges in the localization process, and outlines strategies for mitigating these risks.

1. Risk: Incomplete or Inaccurate String Extraction

- Description:

- There is a risk that some translatable strings may not be accurately identified or extracted during the localization process, leading to incomplete localization or potential errors in the final product.

- Mitigation Strategy:

- The engineer will use carefully crafted regular expressions (regex) to ensure accurate identification of translatable strings. Additionally, manual review of the source code will be conducted to verify that all relevant strings have been extracted.

- A feedback loop with the client will be established during the user testing phase to identify and correct any missed strings before final deployment.

2. Risk: Technical Limitations and Script Failures

- Description:

- The Python scripts used to automate the localization process may encounter technical limitations, such as handling large files, complex code structures, or unforeseen edge cases that could cause script failures or incomplete processing.

- Mitigation Strategy:

- The engineer will develop modular and flexible scripts that can be easily adjusted if issues arise. Initial testing of the scripts on sample files will help identify potential problems early in the process.

- Manual intervention will be planned for handling complex scenarios or edge cases where the scripts may not perform as expected.

3. Risk: Incompatibility with Client’s Build and Deployment Processes

- Description:

- There is a risk that the localized source code and JSON/CSV files may not be fully compatible with the client’s existing build and deployment processes, potentially causing integration issues.

- Mitigation Strategy:

- The engineer will closely collaborate with the client during the development and testing phases to ensure that the localized files are aligned with the client’s technical environment and requirements.

- Compatibility testing will be conducted before the final deployment to verify that the localized files work seamlessly within the client’s build and deployment workflows.

4. Risk: Delays Due to Feedback Integration and Issue Resolution

- Description:

- Delays may occur if significant issues are identified during the user testing phase, requiring additional time for feedback integration and issue resolution.

- Mitigation Strategy:

- A clear feedback loop and communication plan will be established with the client to ensure that any issues are promptly addressed. Time buffers will be included in the project schedule to accommodate potential delays.

- Prioritization of critical issues will be implemented to ensure that essential problems are resolved first, minimizing the impact on the overall timeline.

5. Risk: Resource Constraints

- Description:

- Resource constraints, such as limited availability of personnel or tools, could impact the project’s progress and lead to delays or compromised quality.

- Mitigation Strategy:

- Resource allocation will be carefully planned and monitored throughout the project to ensure that all necessary personnel and tools are available when needed.

- Contingency plans, including backup personnel or alternative tools, will be developed to address any unexpected resource shortages.

Conclusion:

By proactively identifying potential risks and implementing effective mitigation strategies, the project is better equipped to handle challenges that may arise during the localization process. This risk management plan aims to minimize the impact of technical and logistical issues, ensuring that the project stays on track and delivers high-quality results to the client.

- 8.4 Quality Assurance

Quality assurance (QA) is a vital part of the localization project, ensuring that all deliverables meet the highest standards of accuracy, functionality, and consistency. The QA process involves thorough code reviews, validation checks, and collaboration with the client to ensure the final product is error-free and ready for deployment.

1. Code Reviews:

- Peer Review Process:

- The source code, including any scripts written for the localization process, will undergo a peer review process. This involves another software engineer or team member reviewing the code to identify potential issues, ensure adherence to best practices, and verify that the code is well-documented and maintainable.

- The review will focus on the accuracy of string extraction, the correct implementation of localization markers, and the overall integrity of the modified source code.

- Incremental Reviews:

- Code reviews will be conducted incrementally as different sections of the code are completed. This approach allows for early detection and resolution of issues, reducing the likelihood of significant problems arising later in the project.

2. Validation Checks:

- Automated Validation:

- Although the primary focus is on manual processes, some automated validation checks will be employed where applicable. For instance, syntax checking tools may be used to ensure that the JSON and CSV files are correctly formatted and free from errors.

- The scripts used to generate the localized files will also include basic validation logic to check for completeness, such as verifying that all expected strings have been extracted and correctly organized by locale.

- Manual Validation:

- Manual validation will be performed by the engineer and QA Specialist to ensure that all translatable strings have been accurately identified, extracted, and marked within the source code. This process will involve running the localized source code in a test environment to check for any issues that might affect functionality.

- The QA Specialist will also review the translated content to ensure that it is contextually appropriate and correctly integrated into the website's interface.

3. Collaboration with Client:

- Client Feedback:

- A crucial part of the QA process involves collaboration with the client during the user testing phase. The client will be asked to provide feedback on the localized content, including any issues they encounter with the accuracy or functionality of the translations.

- This feedback will be used to make necessary adjustments before the final deployment, ensuring that the localized content meets the client's expectations.

- Final Validation:

- After all feedback has been integrated and issues resolved, a final validation check will be performed to confirm that the project deliverables meet the agreed-upon criteria. This includes verifying that the source code functions correctly, that all strings are accurately localized, and that the content is ready for deployment.

Conclusion:

The quality assurance process is designed to maintain high standards throughout the localization project, with a focus on thorough code reviews, comprehensive validation checks, and close collaboration with the client. By following these QA procedures, the project aims to deliver a final product that is accurate, functional, and fully aligned with the client’s needs.