XML and JSON Translator

Ishup Ali Khan

Haaga-Helia University of Applied Sciences
Master of Business Administration
ICT Services and Systems
Master's Thesis
2025

Abstract

Author(s)
Ishup Ali Khan

Degree
Master of Business Administration

Report/thesis title
XML and JSON Translator

Number of pages and appendix pages
61 + 3

This thesis explores key challenges in localizing structured XML and JSON documents. These challenges were specifically focused on the need to preserve document structure post localization, reduce reliance on localization vendors, and integrate with existing software ecosystems. A custom machine translation application was developed that utilizes the machine translation services to translate structured documents while maintaining structural integrity.

The translator application was tested across multiple European languages. The application was effective in preserving nested XML structures, HTML tags, and placeholders. Qualitative analysis of these translated content showed that translation quality varied by languages complexities. Two translation services Claude API and Hugging Face Helsinki NLP models were used to translate the structured content. Among the two translation services used, Claude API generally produced more contextually appropriate translations in comparison to the translation from Hugging Face Helsinki NLP models. User evaluations highlighted structural preservation and workflow compatibility as the most valuable features of the application.

The implementation leverages a containerized architecture which support flexible deployment across on-premises and cloud environments. This thesis demonstrates that targeted, structure-preserving translation workflows can significantly enhance localization processes by reducing manual intervention and improving consistency. The approach offers a practical alternative to fully human translation and highlights the potential to utilize machine and human translation forming a hybrid workflow in the evolving landscape of language technologies.

Keywords
Agile, Translator, Machine Translation, Localization

Table of contents

1 Introduction

In the contemporary global business landscape, efficient content localization has evolved from a competitive advantage to an essential strategic requirement. Research shows that 76% of consumers prefer purchasing products with information in their native language (CSA Research. (2020). Can't Read, Won't Buy: B2C (Summary) s.a.). Localized marketing campaigns generate up to 197% higher conversion rates than their single-language counterparts (2025 Marketing Statistics, Trends & Data s.a.).

The technical foundation of modern digital content relies heavily on structured data formats, primarily Extensible Markup Language (XML), JavaScript Object Notation (JSON), Structured Query Language (SQL), Comma-Separated Value (CSV), etcetera. The separation of content from presentation makes these formats industry standards for organizing information across platforms and languages (Working with JSON - Learn Web Development | MDN, n.d.).

This thesis presents the XML and JSON Translator application, a targeted solution aimed at improving the efficiency of translating structured data files in a company-specific context. While larger localization systems for such files exist, this application improves how organizations can manage and implement the process to incorporate it among their own existing technology, maintaining technical accuracy.

1.1 Background and Context of Localization Challenges

Modern businesses face increasing pressure to deliver localized digital experiences. According to the article by McKinsey & Company (Revenue growth: Ten rules for success | McKinsey s.a.) , organizations that expand internationally generated 1.9 percentage points more annual Total Shareholder Return (TSR) than their industry peers. This clearly indicates the need of comprehensive localization strategies to grow in international markets. For technology companies, this represents both an opportunity and a significant operational challenge.

Due to the structured nature of XML and JSON formats, where translatable contents are embedded within the technical syntax that must be preserved, translating these formats poses unique challenges in the localization process. Some of these challenges include maintaining the structural integrity of tags and keys while isolating localizable content, which can complicate both automated and manual translation workflows (Esselink 2000). The following are challenges in these formats:

1. **Structural Integrity:** These formats have strict syntactical requirements, so any structural changes make files unusable.

2. **Mixed Content:** These formats contain combinations of translatable text and non-translatable code elements that must remain unaltered during translation (XML introduction - XML: Extensible Markup Language | MDN s.a.).

3. **Special Sections:** Some of the elements in XML documents, such as CDATA sections, require specialized handling to preserve their structure while translating content (Allen 2006).

4. **Context Limitations:** Some contextual information is lost during text extraction, which could lead to inaccurate translations(Esselink 2000).

5. **Volume Challenges**: Enterprise applications frequently consist of thousands of text strings across hundreds of files making manual processing impractical (Usability 101: Introduction to Usability - NN/g s.a.).

Some of these technical difficulties cross with commercial needs:

- **Time-to-Market Constraints:** While traditional localization processes often span weeks or months, global product launches require simultaneous availability across languages (De La Cova 2016).

- **Cost Pressures:** Professional translation services typically charge between $0.15-0.25 per word for technical content, hence incurring significant expenses at enterprise scale (Slator*2023 Language Industry Market Report - Slator, n.d.)*.

- **Quality Requirements:** Substandard translations negatively impact brand perception and user experience. According to Gartner research (Gartner(Gartner Inc. (via Public) / Gartner Survey Finds 62% of Customer Service Channel Transitions are “High Effort” s.a.), 74% of users are likely to abandon applications with inadequate localization.

- **Maintenance Burden:** Content updates require continuous localization, which increases operational overhead with each supported language (Esselink 2000).

Some of the approaches to these challenges like manual translation, general translation Application Programming Interface (API)s, enterprise translation management systems, and custom scripts, each present significant limitations in handling structured data formats efficiently (Höppner, Haas, Tichy & Juhnke 2022; Key vs file-based translation management systems (TMS): challenges and benefits - Lokalise Blog s.a.; Limitations of Translating Database Content with Machine Translation [2025] s.a.).

1.2 Problem Statement and Market Opportunity

This thesis addresses the following fundamental problem:

Organizations developing multilingual software application face several interrelated challenges in translating XML and JSON documents. Existing translation service providers in the market often fail to maintain the structural integrity of these documents, resulting in the broken functionality within the applications. Dependencies on specialized localization vendors increases costs and delays in software release cycles. Integrating these third-party services with the current ecosystem of the company would pose additional challenges. This integration is impeded by a variety of factors, including the security and proprietary nature of the content, which are required by the unique technologies and architectures of different companies. Additionally, although professional human translations offer high quality, it demands significant financial and time investments. These challenges altogether, hinder effective internationalisation and highlight the need of a solution that preserves structure integrity, seamlessly integrates with existing development environment of the company, reduces reliance on external vendors, and achieves a practical balance between quality, cost, and speed.

This is not a problem specific to a single company. Software companies developing user interfaces, e-commerce businesses with product catalogues across multiple regions, and content management providers all face this challenge daily (Hashimoto & al. 2020).

Market analysis reveals the scale of opportunity. The global language services and technology market reached $26.6 billion in 2022 and is projected to grow (Slator 2023 Language Industry Market Report - Slator s.a.). While structured content translation represents a specialized segment within this market, its experiencing accelerated growth driven by digital transformation initiatives. There are five key factors driving this growth: companies accelerating digital transformation initiatives, businesses expanding global reach, increasing regulatory requirements for localized content, more agile development practices requiring faster translation cycles, and persistent operational cost pressure forcing efficiency improvements (CSA Research. (2020). Can’t Read, Won’t Buy: B2C (Summary) s.a.).

The market opportunity lies in creating real business value through automated translation of structured data formats without compromising technical integrity. A built-in localization system would cut down localization timeframes from weeks to hours, significantly reducing translation costs. Moreover, a purpose-built solution addressing these pain points would transform how businesses approach multilingual content management, freeing technical teams from translation-related maintenance and accelerating global product deployment.

1.3 Research Questions and Objectives

This thesis addresses four primary research questions:

1. How can machine learning models be tuned to preserve format integrity while translating ordered data?
2. In comparison to general-purpose solutions, what technical and financial benefits exist from a specialized XML/JSON translating tool?
3. How do performance, cost, and quality change depend on the translation engine—open-source or proprietary AI models?
4. How much would automating structured data translation have a measurable commercial influence?

The key objectives of this project are to:

1. Design and implement a specialized translation tool for XML and JSON files that preserves structural integrity.
2. Evaluate the performance of various machine learning translation approaches.
3. Develop a scalable architecture suitable for personal and business usage.
4. Design a user-friendly interface to streamline the process's technical intricacy.
5. Evaluate the business value and possible return on investment.

1.4 Project Scope and Limitations

This project includes:

- Complete application development with backend (Python/FastAPI) and frontend (Next.js).
- Support for XML and JSON file formats with preservation of structure, IDs, and special elements.
- Integration with both free (Hugging Face) and premium (AWS Bedrock/Claude) translation engines.
- Implementation of containerized deployment for cloud and on-premises scenarios.
- Potential business impact analysis.

The project has some limitations:

- Translation quality and time taken for the translation are dependent on the underlying machine learning models and their training data.
- Initial development language support is limited to the availability of languages in the Helsinki-NLP models.
- The solution focuses on English as the source language for the initial implementation.
- Performance benchmarks are based on limited test scenarios and may vary in production environments.

1.5 Thesis Structure and Methodology

This thesis combines technical development with business analysis through a thorough structure that examines the solution from the viewpoint of technological implementation, market positioning, financial viability, and practical deployment.

The methodology employs both software engineering practices and business analysis techniques, including iterative development cycles with continuous testing, user-centred design based on research with potential users, empirical testing of translation quality and performance, and competitive analysis and market research.

1.6 Use of Artificial Intelligence

This thesis has utilized artificial intelligence tools including Claude AI, ChatGPT, and Cursor AI to support the development and structuring of ideas, as well as to assist in drafting an English-language questionnaire. AI-generated assistance was used iteratively throughout the writing, and application development process including in tasks such as refining research questions, for example: "How can the methodology section of the thesis be further delimited and broken down into sub-objectives?". “How to implement lazy loading of the Hugging face model to reduce application building time?”. Additionally, AI tool was also used while creating a test corpus formatted specific to the application need using the prompt, “Covert the following ApplicationTexts.xml file into JSON format”. Cursor AI tool/editor was used in development assistance, code completion, troubleshooting and debugging.

While AI tools contributed to generating preliminary drafts and exploring alternative phrasings, all content has been carefully reviewed, edited, and refined by the author to ensure clarity, accuracy,

and relevance. The final text reflects the author's own critical thinking, academic judgement, and writing.

These tools were employed responsibly, with due consideration to data protection, intellectual property, and academic integrity. All sources cited in this thesis are properly referenced, based on verifiable and credible materials, and no AI-generated citations have been included.

2 Literature Review

This section goes over examining the theoretical and practical context in which the XML and JSON Translator application operates. In this review, three related domains that influence the approach of the project were investigated. First, present trends in corporate globalization and localization, stressing operational difficulties and economic factors confronting companies aiming at multilingual content strategies, were analysed. Then the state of machine translation technologically, emphasizing developments in structured data handling techniques and neural network architectures, was evaluated. Finally, the dynamics of translating solutions in the market to find chances for competitive positioning and unmet needs among different client groups was examined. With the help of this multifarious study, the technical framework and business case that direct the evolution of our specific translating solution were established.

2.1 Business Globalization and Localization Trends

The speed of digital transformation has fundamentally changed how companies approach world markets. Businesses are realizing localization not only as a technical difficulty but also as a strategic need directly influencing customer experience and market penetration (Esselink 2000). Globalization helps companies to reach more people, but good localization guarantees that products appeal to local markets.

According to recent Common Sense Advisory research, 40% of consumers refuse to buy at all from websites in other languages, and 76% of consumers prefer buying goods with information in their native tongue (CSA Research. (2020). Can’t Read, Won’t Buy: B2C (Summary), n.d.). This consumer preference puts real economic pressure on companies to apply thorough localization plans.

Conventional localization methods continue to be expensive and labour-intensive. With costs accelerating as content volume increases, according to the data presented in Unbabel’s 2023 Global Trends in Marketing Localization Report businesses usually allocate between 1% and 4% of their overall operating budget to translate and localize activities (The Business Impact of Translation and Localization - Unbabel, n.d.). These costs mostly come from human translation services, which, despite their quality, cause major obstacles in the paths of product deployment.

The financial ramifications go beyond direct translation expenses. Delayed market entry brought on by localization bottlenecks can greatly affect revenue potential (Herm, 2013). Often, the opportunity cost of this exceeds the direct translation costs.

Enterprise localization processes present several ongoing difficulties. For preserving both linguistic accuracy and technical functionality, content structured in formats like XML and JSON are used which requires specialized handling. These formats are commonly used in software programs, e-commerce platforms, and content management systems. In addition, the growing acceptance of continuous deployment techniques strains conventional localization processes not meant for fast iterative cycles.

2.2 Technological Assessment

From early rule-based systems to statistical methods and, more recently, neural network architectures, the machine translation terrain has changed significantly. Utilizing encoder-decoder architectures with attention mechanisms that greatly raise translation quality, modern neural machine translation (NMT) marks a substantial breakthrough (Vaswani & al. s.a.). These models generate more natural-sounding translations and better capture contextual relationships than previous approaches.

Open-source NMT systems have made advanced translating capabilities widely available. The OPUS-MT models of the Helsinki-NLP project, maintained by the University of Helsinki Language Technology research group, provide specialized models for hundreds of language pairs. These models, which are accessible to developers without business budgets and offered on sites like Hugging Face, offer reasonable translating quality.

More complex models are available for commercial uses via cloud providers. Particularly for specialized content (Build Generative AI Applications with Foundation Models – Amazon Bedrock – AWS, n.d.), AWS Bedrock provides access to foundation models from companies like Anthropic, Cohere, Meta, Mistral AI, Stability AI, and Amazon itself. These models show exceptional handling of difficult translating chores. Though at significantly more cost, these models often outperform open-source alternatives in specialized fields.

Translating structured documents presents technical difficulties beyond linguistic accuracy. JSON and XML contain both machine-interpreted structural elements and human-readable content. Translation systems must carefully preserve these structural elements while only modifying appropriate textual content. Focusing on this field, (Yves Savourel's, 2020) explains how often naive methods of structured document translating compromise document integrity, leading to downstream system failures.

Structured data translation raises particular questions with regard to placeholder management, formatting tag preservation, and handling of special sections like CDATA in XML. Industry

standards like XLIFF (XML Localization Interchange File Format) attempt to address these issues but introduce additional complexity to implementation workflows (XLIFF Version 2.1, n.d.).

2.3 Market Analysis

The global language services market reached $62.6 billion in 2022 and is expected to expand at a CAGR of 7.9% through 2027 (LanguageServices Market Global | Industry Analysis, Size & Trends Report, n.d.). Driven by the expansion of software localization requirements and API-first development approaches, specialized translating tools for structured data constitute a growing niche within this more general market.

There are three main categories that define current market solutions: general-purpose translation APIs (such as Google Translate or DeepL)(Translate documents | Cloud Translation | Google Cloud s.a.), complete enterprise localization systems (such as Smartling or Phrase)(How to Understand Localization Costs: A Detailed Guide s.a.), and specialized developer tools. General APIs are simple, but they lack the structural awareness required for sophisticated document formats. Thorough processes are offered by enterprise platforms; however, enterprise systems come with a lot of implementation overhead and premium pricing. Developer tools provide more accuracy, but their effective deployment usually depends on a great degree of technical expertise.

A 2023 Global App Testing survey (Report*: State of Localization Quality in 2023, n.d.)* revealed that 69% of localization experts ranked cross-functional collaboration as one of their top three internal challenges. This shows the significant friction between localization and technical teams. Furthermore, 58% of respondents said that speed influences quality, implying that present localization solutions might not be ideal for handling structured data formats, leading to delays and higher deployment pipeline expenses.

Pricing sensitivity differs greatly among market sectors. Enterprise customers give quality and integration capabilities top priority, with typical budgets of $100,000+ annually for localization technology. Usually allocating $25,000–75,000 annually, mid-market businesses look for reasonably priced solutions that balance cost with quality. Small businesses and independent developers want pay-as-you-go models with low upfront investment (Slator 2023 Language Industry Market Report - Slator s.a.) therefore, they require accessible options.

The competitive landscape reveals a clear difference between developer-oriented tools that give technological flexibility but lack user-friendly interfaces and business-oriented solutions that offer complete capabilities at premium prices. Few solutions successfully close this gap with a method combining technical accuracy, simplicity of use, and economy of cost.

3 Methodology

In this section, the research design, technical approaches, and evaluation techniques used in the development of the XML and JSON Translator application are described. This approach combines empirical evaluation with software engineering techniques, ensuring both technical rigor and practical utility. The mixed-methods research framework that guided design decisions, the iterative cycle development process, testing and evaluation of the implementation, and underlying limitations of the project are discussed in this section.

3.1 Research Approach

Prior to the development of application certain research approaches were identified as appropriate for the project development. The project focuses on both technical and business-oriented approach, therefore following strategies were considered.

3.1.1 Mixed-Methods Framework

This project employed a mixed-method research approach by combining both quantitative and qualitative research techniques to develop a comprehensive understanding of problem domain. This methodological triangulation gave multiple perspectives on the difficulties of structured document translation, thus enabling more complex insights than would be possible with a single method framework.

The research's quantitative elements comprised:

- Comparative analysis of metrics of translation quality among two different language models.
- Performance benchmarking of processing times under different document complexity settings like size and formats.
- Analysis of structural preservation accuracy across the translated document formats.

The qualitative elements consisted of:

- Content and process analysis of current translating systems and their limitations.
- Heuristic analysis of current market solutions to find usability constraints.
- Real case evaluation of localisation issues in contexts of software development. Example: compulsory process flow, dependency on human translator knowledge and availability and delays in delivery timeline.

This mixed-methods approach followed the convergent parallel design described by Creswell and Plano Clark (Creswell & Clark 2017), whereby quantitative and qualitative data were gathered

concurrently, analysed separately, and then merged to produce a more comprehensive understanding of the problem space. Combining these approaches helped in the identification of important needs possibly overlooked in a single-method approach. Figure 1illustrates the convergent parallel mixed method applied to this project.

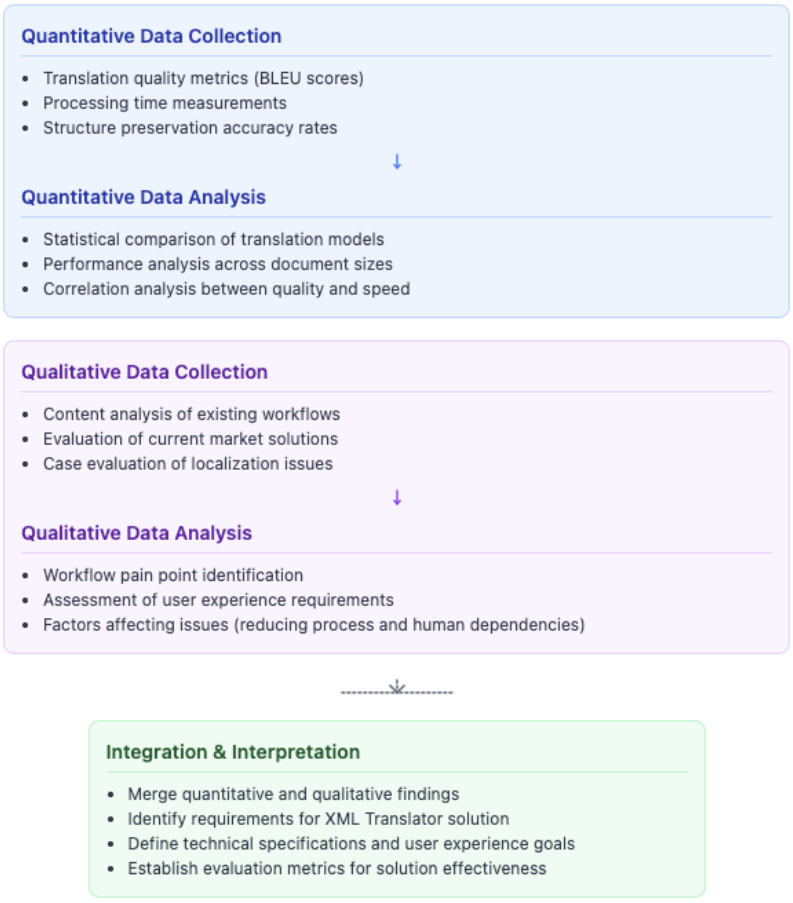


Figure 1: Convergent Parallel Mixed Methods Design applied to the project based on (Creswell & Clark 2017)

Given the specialized nature of the problem domain, the methodology prioritized developing a working prototype that could demonstrate the viability of the proposed solution. This prototype-driven approach made it possible to iteratively improve based on testing results and practically evaluate technical ideas.

3.1.2 User-centred Design Principles

Throughout the research and implementation process, the development approach incorporated ideas of user-centred design. The design process maintained explicit understanding

of users, tasks, and environment, hence also involved actual user of the tool, localisation specialists to advice on design and development.

User research was conducted through contextual inquiry sessions involving some localisation specialists. These sessions provided understanding of the practical challenges with current localisation processes including:

- Workflow bottlenecks created by various documents processing including quote approval, payment and date agreements, translator management etcetera.
- Quality control challenges when some placeholders not meant for translation were translated, and fonts and orientations of some texts were changed unintentionally.
- Complete reliance on the localisation vendor for validation. Time limits preventing complete validation of translated materials.

Therefore, the personas created from this research directed later design decisions such that the application satisfied the requirements of the primary user groups. This user-centred approach ensured that the technical solution remained grounded in practical needs.

3.1.3 Iterative prototyping methodology

The research project used an iterative prototyping technique that allowed for progressive refinement of both the conceptual model and technical implementation. Following the spiral model put forth by Boehm ((Boehm 1988), the development process passed through various cycles of determining objectives and constraints, evaluating alternatives, and identifying risks, developing, and verifying the solution, full integration testing and planning the next iteration. This approach was useful provided the complex document structure, translation models and user interface requirements.

Prior to the actual development, two separate but simple applications were developed to test with one liner texts and elements. The core functionalities of the main application were iteratively fine-tuned during this early development. This proof of concept was tested against couple of languages with both Hugging Face model as well as Claude API, to make sure the core functionalities were functional. Later versions included more refined capabilities, including batch processing, two different file formats support, placeholder preservation and better user interface.

Feedbacks were collected following every major iteration to enable course corrections before significant development resources were committed. This method avoided a common mistake of creating technically sound but practically unusable solutions that exists in specialised software development.

3.2 Technical Implementation Framework

This application was developed using a modern stack emphasizing on modularity, scalability, and developer experience. The language and frameworks selection, documents format consideration, language models integration and AI ecosystem involvement are discussed in this section.

3.2.1 Language and Framework Selection:

The technical implementation required careful selection of programming languages and framework for flexible deployment and better performance. The backend system was developed using python 3.10. The programming language was chosen for its robust ecosystem of natural language processing libraries and strong support for asynchronous programming patterns. The web framework FastAPI was selected based on several advantages like better performance characteristics than traditional python framework (30 Days Coding s.a.), simplifies integration with other systems, support for asynchronous request handling, middleware support for cross-origin resource sharing (CORS) and authentication.

The frontend implementation was done with a react framework, Next.js (Next js Tutorial: A Complete Guide to Building Fast and Scalable Web Apps s.a.). Some of the benefits of this framework are server-side rendering capabilities for better page loading performance, deployment flexibility, simplified backend communication due to built-in API routes, responsive user-interface.

This combination of efficient technologies was validated through performance testing which confirmed the anticipated workload was handled with acceptable response time.

3.2.2 Document Format Consideration

This application was designed to support structured document formats like XML and JSON for software localisation and content management systems. While JSON is preferred for its simplicity and fit with web apps, XML is widely used in Android apps and legacy systems(About localization files and how they’re used - POEditor Blog s.a.).

Some of the challenges for translating XML file formats are:

- XMLs contain mixed content models combining markup with text
- Special structures like CDATA sections in XML files require preservation.
- Namespaces in XML files must be maintained for document validity
- Combination of both attributes that may or may not require translation.

Additionally, JSON formats present different considerations:

- Correct traversing of nested object structures
- Array handling that requires positional awareness.
- No set method for differentiating translatable and non-translatable strings
- Insufficient metadata to direct translations

Hence, due to such complexities specialised parser for each format handling was required. The implemented parser accurately identifies translatable content, extract this specific content while preserving structural context, passes only the translatable content to the translation service and later re-inserts the translated content maintaining the structure. During this implementation, form integrity was given top priority over processing speed. This ensures that the produced documents would remain functionally equivalent to their original versions and technically valid.

3.2.3 Integration of Language models

This application was designed to use both open-source and commercial language models for translation services and compare the performances and quality of the translation generated from these models. This dual approach required building abstraction layers allowing interaction with different APIs while presenting a consistent interface to the rest of the application .

For open-source model, Hugging Face's Transformers library, specifically the Helsinki-NLP OPUS-MT models were used (Tiedemann & al. 2022). These models were selected based on:

- Permissive licencing allowing for commercial use.
- Reasonable computational needs allowing deployment on small-sized hardware
- Active maintenance combined with community support

Whereas premium version of Claude Sonnet 3.5 Sonnet model from Anthropic (Introducing Claude 3.5 Sonnet \ Anthropic s.a.) was used as a comparison model. High context awareness, web-scale dataset, better reasoning knowledge, cloud-based availability etc are some of the features considered while selecting the paid model.

Based on user preference, available credentials, and particular language pair needs, the system factory design pattern design let runtime selection among these services.

The whole technical architecture was designed keeping in mind three core requirements:

- Document structure preservation, including special elements (placeholders, CDATA, etcetera.)
- Flexibility to support two document styles (XML and JSON)
- The ability to scale and manage batch processing of multiple files.

With separate modules for document parsing, translating service integration, and document reconstruction, the implementation strategy gave top priority to clear separation of concerns. This modular approach improved testability and would enable future extensions to support additional file formats or translating tools.

3.3 Testing and Evaluation Framework

The evaluation framework included qualitative evaluation of translation quality as well as technical performance criteria. This section discusses on testing strategy used, methods utilised in quality assessments and samples development for testing and validation.

3.3.1 Comprehensive Testing Strategy

The evaluation framework assured both technical accuracy and practical utility by using a multi-layered testing approach. This approach utilised the traditional software testing methodologies for evaluating the machine translation systems. Following testing methods were included in the testing strategy (Anwar & Kar 2019):

Unit testing: This testing focused on testing individual components including document parsers, translators, and structural preservation techniques and verifies that each component functioned correctly in isolation.

Integration testing: This testing evaluated the interaction between multiple components functioning together especially the handoff between document processing and translation services. These tests were done to ensure that data moved across the system in order.

End-to-End testing: This testing verified that the entire system was functioning properly when a complete application flow was simulated.

Performance testing: Potential bottlenecks and scaling limits were tested by introducing various load conditions like batch files processing and large document processing.

Cross-format testing: During this testing documents with different formats were introduced at once for translation. These tests ensured the system could handle complexly structured real-world documentation.

From low-level parsing errors to user experience concerns, this comprehensive testing strategy helped find and fix issues at multiple levels of the system.

3.3.2 Quality Assessment Methodology

Maintaining the quality of the translation presents unique challenges unlike those of standard software quality metrics. Both technical accuracy and linguistic quality should be maintained during the translation process (Petrova 2019), hence the evaluation framework keep these requirements into consideration such that the structure of the documents is preserved, and the translations produced were appropriate.

For technical accuracy the assessment included structure validation, content verification, placeholder integrity and special section handling like CDATA.

Whereas for linguistic quality, the evaluation used reference-based metrics comparing the translation with some of the human translated content and comparative evaluation with some other translation tool like google translate.

The quality assessment methodology acknowledged that quality thresholds depend on the different use cases. User interface localization require more precision and accurate translation in comparison to the marketing content which prioritised natural expressions.

3.3.3 Test Corpus Development

During the early development phase, some representative test documents were created containing texts embedded inside some xml tags. This was done to make sure the core functionality of the application was functioning. Later, as the application included complex algorithms, a test corpus was developed to ensure the evaluation covered realistic use cases. The test corpus contained user interface strings, documentation segments, and mixed formatted texts.

The corpus was built to incorporate different degrees of complexity. The content included simple, loosely structured documents, nested components, long texts, and placeholders. This stratified approach ensured that evaluation results would generalise to the diverse documents used in the real world. A code snippet from the document content created as a test corpus is shown below.

```
<?xml version="1.0" encoding="utf-8"?>

<LOCALIZATION version="1.0" id="en" name="English">

  <TEXT id="activate.placeHolderText">Please enter code</TEXT>

  <TEXT id="authenticate.locked.message"><![CDATA[<p>For security
reasons, authentication is currently unavailable.</p><p>Please try again
later </p>]]></TEXT> </LOCALIZATION>
```

3.4 Implementation Process

The implementation comprised four primary phases of the iterative development approach. These phases include Market research data collection and analysis, Agile development methodology and Backend and frontend implementation.

3.4.1 Market research, data collection and analysis

Prior to the technical implementation for the project was initiated, comprehensive market research was done to understand the competitive landscape and identify opportunity. Some of the localisation service providers like Smartling, Phrase, Lokalise, Transifex, Lionbridge and Global Link String were taken into consideration as a part of research analysis based on the pricing models and the localization approaches these vendors are providing. Data recorded from these vendors were compared and analysed as a part of the data recording framework. Additionally, further data collection and analysis were performed on the basis of the turnaround time and the validation of the delivered translation content. For the later part, only the validation timing from the two primary vendors Lionbridge and Global Link String (TransPerfect) were considered. Table 1 below shows the summary of the vendors, their pricing models, and key features.

Table 1: Summary of localization vendors, Pricing model and Key features

Vendor	Pricing Model	Key Features
Smartling(Smartling vs. Lokalise | How Do They Compare? s.a.)	Tiered plans with flexible purchasing options	TMS with integrated language services, visual context, customizable workflows
Phrase(Introducing Our New Subscription Pricing | Phrase s.a.)	Unified platform pricing	Comprehensive suite with TMS, AI translation, analytics, and integrations
Lokalise(Lokalise Pricing 2025 s.a.)	Tiered plans based on features and usage	Cloud-based TMS, collaboration tools, mobile SDKs, extensive integrations
Transifex(Transifex Pricing | Plans that scale with your localization needs s.a.)	Tiered plans with add-on services	Continuous localization, automation, visual context, API access
Lionbridge(Gengo – Professional Translation Services Within Hours s.a.)	Custom pricing	Full-service language solutions, AI tools, industry-specific services
GlobalLink(Pricing • GlobalLink Strings Software Localization s.a.)	Custom pricing	Enterprise-grade TMS, workflow automation, CMS integrations

Additionally, analysis was done based on the pricing tiers, average turnaround time and supported languages among the vendors, the visualization of the result is shown in the Figure 2.

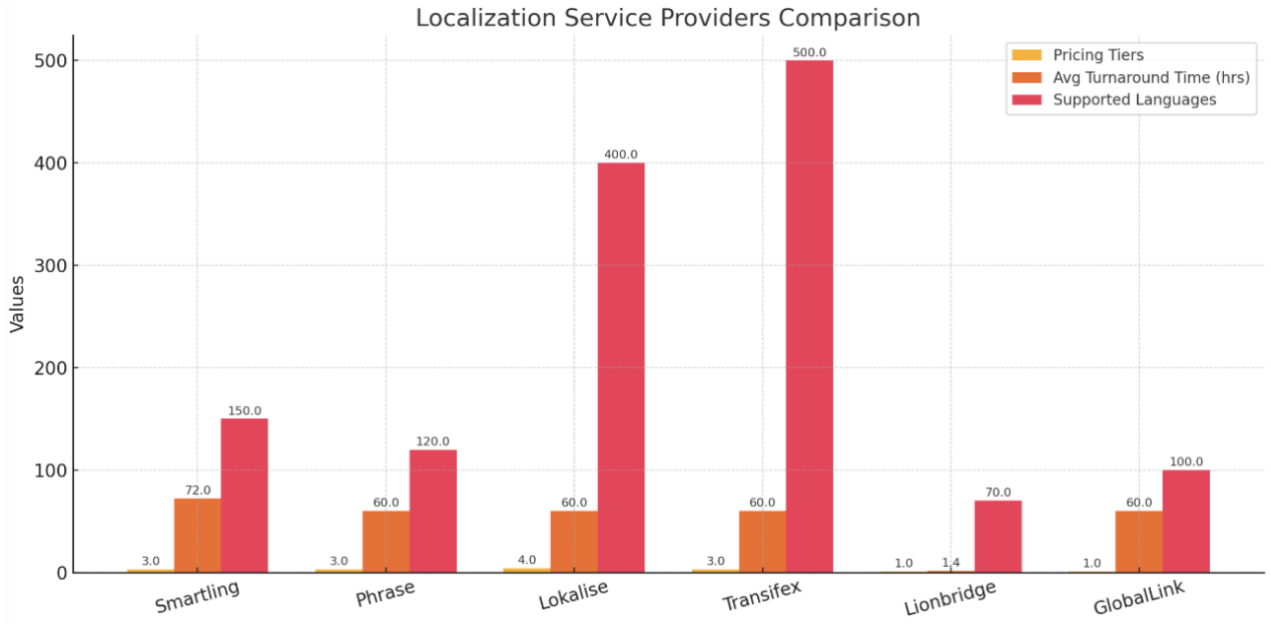


Figure 2: Localization service providers comparison for pricing, average turnaround time and supported languages.

In particular pricing policies and the range of supported languages, this market research and data analysis of localisation service providers delivered insightful analysis of the key competitive elements influencing the sector. Especially, the study revealed a lack of explicit consideration for document complexity and turnaround time in publicly accessible vendor comparisons. Customised business plans tailored to particular client requirements, likely handle these elements. Moreover, the findings serve as a reference for the development of an in-house, customisable translating tool able to solve limitations of current vendor solutions.

3.4.2 Agile Development Methodology

The project was implemented using Agile methodology principles, more especially with Scrum as the development management framework. This method was chosen for its ability to meet changing needs and enable regular feedback inclusion, iterative testing, planning and development, thus practicing the values and principles of Agile, as outlined in Agile Manifesto (Manifesto for Agile Software Development s.a.). Although Agile approaches are traditionally designed for and most commonly in collaborative, multi-member teams (The Scrum Guide s.a.), this section looks into their adaption for a single-developer environment. It describes the particular frameworks and methods used and assesses the results obtained using this tailored but iterative, principle-driven development approach as depicted in Figure 3.

Figure 3: Agile Model (Agile Process Diagram | EdrawMax | EdrawMax Templates s.a.)

Originally developed as team-based methods for software development, agile techniques stress iterative development, customer feedback, and teamwork. Usually assuming multi-person teams with clear roles (The Scrum Guide s.a.) and (Beck s.a.). Research by (Dybå, Dingsoyr & Moe 2014) shows, however, with appropriate adaptation, Agile values and principles can be quite fit for solo development environments.

For this project, what ((PDF) Personal Extreme Programming–An Agile Process for Autonomous Developers s.a.) refer to as "Personal Agile," was adopted which preserves fundamental Agile ideas while modifying practices to fit individual progress. This strategy fits the results of (Murphy-Hill & al. 2021), who observed 76% of surveyed solo developers reported utilising modified Agile approaches instead of rigorously following any single framework.

This adaptation preserved four fundamental values from Agile Manifesto:

1. "Individual and team interactions over processes and tools
2. working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan"

Working alone eliminated some team dynamics, but it highlighted the value of other Agile ideas. This solo development needed explicit work organisation and regular self-assessment that might otherwise develop organically from team dynamics.

Instead of standard two-week time-boxed sprints as standard scrum (The Scrum Guide s.a.)recommended, this development project followed one-week time-boxed sprint duration. Based on the project requirements this time period seems to be suitable for maintaining momentum and allowing optimal time for significant increases in capability and testing.

Each sprint followed a modified plan including:

- Sprint planning: Setting particular goals and approximating workload capacity
- Development Period: Using top priorities and tackling technical difficulties
- Testing and Integration: Ensuring consistent operation and verifying capability
- Sprint review: Reviewing completed tasks against objectives.
- Sprint Retrospective: This ceremony was theoretically skipped rather was used to debug some issues.

The framework somewhat aligns from (Kamei, Pinto, Cartaxo & Vasconcelos 2017), who recommended shorter one-week sprints for solo developers to keep momentum.

Solo development presented unique difficulties for time management. Working in concentrated 25-minute intervals followed by brief breaks, a modified version of the Pomodoro Technique as advised by (Cirillo 2018)was used. This method is in line with results from Kearns and Gardener (Kearns & Gardiner 2007), whose studies showed better output and lower procrastination by means of methodical time-boxing strategies.

Backlog management and prioritization: A product backlog with a prioritized list of all desired features and technical needs helped to manage project requirements. The backlog was kept on a digital Kanban board using Trello. Since, it was a new project, the main unit of work was user stories, however as the project progressed various bugs were also added to be fixed based on the priority. As described by Clegg and Barker's (Clegg & Barker 1994) MoSCoW approach, prioritization of the items followed the MoSCoW method (must have, should have, could have, won't have). Figure 4 illustrates the summary of the items and their statuses.

During the development stages there were three main types of work that surfaced, each of which requiring different prioritization approaches.

- Core functionality items: These are the features essential for the basic operation of the application.
- Items for quality enhancement: These items improved reliability, performance, and security.
- User experience items: Features improving usability and interface design.

This classification fits the "three horizons" theory put forth by (Paasivaara, Lassenius, Heikkilä, Dikert & Engblom 2013), which separates immediate needs from longer-term improvements providing backlog clarity.

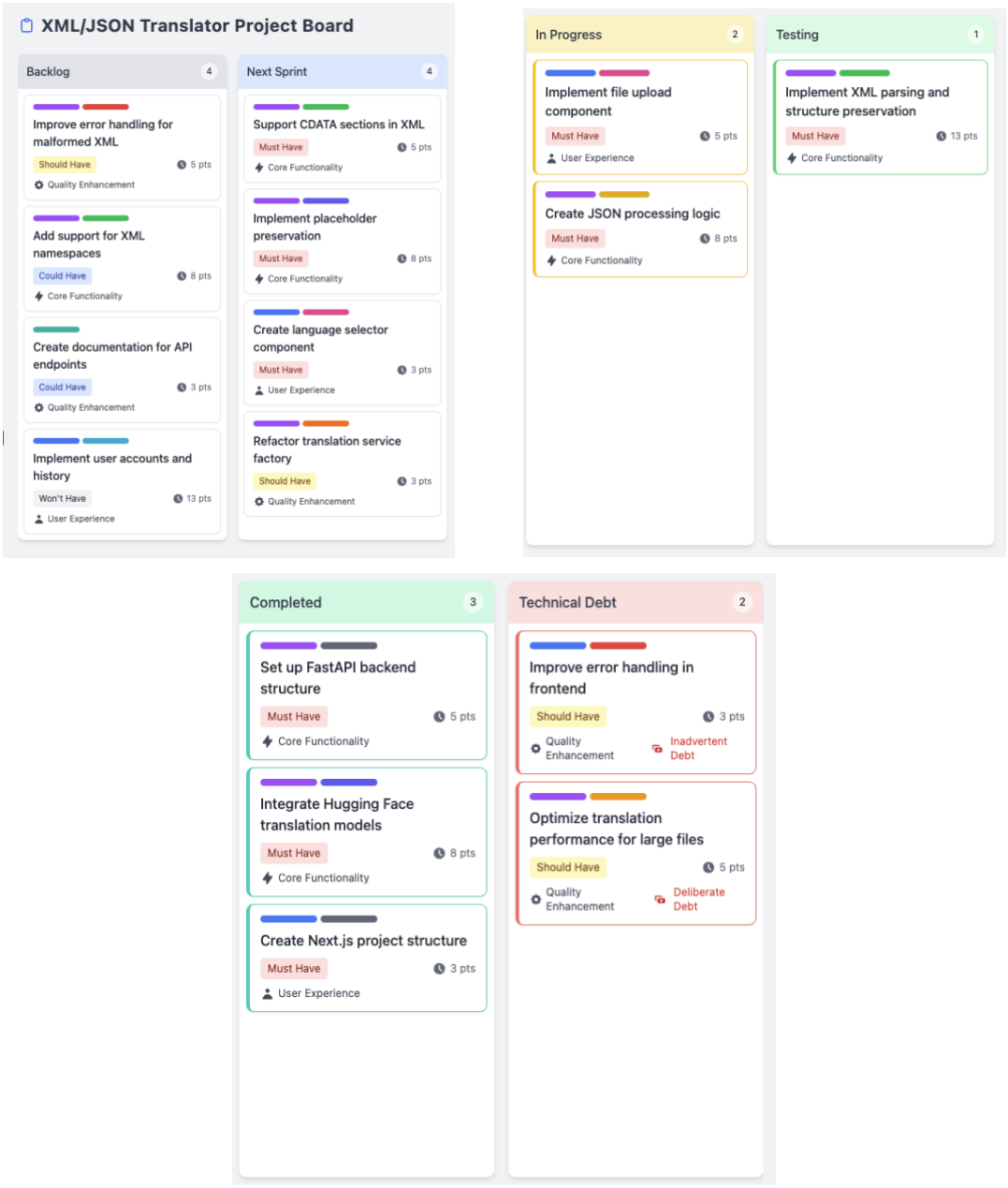


Figure 4:Trello board with items summary and status

Iterative refactoring and Test-Driven Development (TDD): In order to improve code quality and future maintainability, refactoring of code was done amid development once a core functionality was implemented and tested. Refactoring included removing code duplication, improving naming and documentation, simplifying complex functions, enhancing error handling and proper code commenting. This approach is consistent with results of (Kim, Zimmermann & Nagappan 2012), who found that, especially in projects with evolving requirements, regular refactoring is a crucial practice for maintaining development velocity over time. The development approach followed the Test-Driven Development (TDD) for core components. This method proved particularly useful for

the processing of XML and JSON documents since these components needed precise behavior under several edge cases. However, TDD was selectively applied rather than implementing universally. For example, front end components were generally developed using a more exploratory approach with test written only after initial implementation.

Documentation and code management: For this project, documentation served mainly two purposes, design clarity and knowledge preservation. Following forms of documentation were kept constant throughout development:

- Documenting codes: Docstrings and comments explaining details of implementation.
- Architecture documentation: System structure diagrams and explanations, which is a part of this report.
- Decision logs: Records of significant technical decisions and their rationales were documented.
- Code management and version control: GitHub repository was used for managing any code changes and maintaining version control.

Hence, the application of modified Agile approaches for this development project showed that fundamental Agile ideas can be effectively applied to personal work environments.

3.4.3 Backend and Frontend Implementation Plan

The implementation plan of the project enables parallel integration of backend and frontend tracks. This approach allowed simultaneous progress of the components effectively working together.

Backend implementation included the following phases:

1. Core Architecture: Development of the fundamental application structure, configuration management, and API definition.
2. Document processing: XML and JSON parsing logic applied with structure preservation capabilities.
3. Translation Services: Hugging Face models and Claude API integration.
4. Testing framework: Creation of automated tests for validation of functionality.
5. Performance Optimization: Identification and resolution of performance constraints.
6. Deployment configuration: Setting up Docker containers for consistent deployment.

Frontend Implementation included the following phases:

1. User Interface design: Development of application visual design and wireframes using Figma.
2. Component Development: Implementation of reusable UI components for file handling, language choice, and result presentation.
3. State Management: Client-side state management's development for monitoring translation progress.
4. API Integration: Connection to backend service for processing of the document.

After the development completion of every main component, integration points were created to enable early identification of compatibility problems. This strategy reduced the possibility of major late in-development rework.

For every phase, the implementation plans included particular benchmarks and acceptance criteria that clearly set goals for quality evaluation and development advancement.

3.5 Limitations and Assumptions

During the course of project, there were some limitations that shaped the scope of the project. These limitations can be categorized into three primary areas: Scope limitations, Technical Assumptions and Translation quality considerations.

3.5.1 Scope limitations

This research and its implementation followed particular scope limits that should be acknowledged while interpreting the results:

- Language coverage: Language support range for the project varies for the translation models selected. The premium Claude API model has effectively unlimited language coverage. However, only a subset of languages was included for better performance of the application. Whereas, Hugging face model language constraints are based on the pre-loaded language model within the application. Since downloading all available language models is resource intensive and affect the application performance, only selected languages were integrated to balance the functionality with system efficiency.
- Document complexity: The logic to handle the document complexities in the application is limited to the test corpus created based on the real use cases. Documents with unfamiliar and advanced complexities would require extra processing logic beyond the current implementation.

- Content domains: Testing primarily utilises the test corpus developed under categories like software interfaces, marketing materials and technical documentation. Legal or medical content in particular would require for domain-specific models for best translational quality.
- Scale limitations: Performance testing focused on document including 1000 strings per file. Very large documents would require for batch processing or additional optimisation approaches.

These scope restrictions were specifically selected to ensure complete coverage of the most commonly used cases rather than trying more general but shallow implementation.

3.5.2 Technical Assumptions

The project implementation is based on several important technical presumptions:

- Well-formed input documents: The system presuming input documents follow XML or JSON standards. Although fundamental validation is performed, the system is not meant to fix malformed documents.
- UTF-8 encoding: All document processing uses the most often used encoding for web applications, Unicode-8. Other encodings require pre-processing.
- Deterministic document structure: The implementation presumes that translatable content can be consistently found by structural patterns and that document structure is deterministic. Dynamically determined translatability document would require extra metadata.
- Internet connectivity: Using cloud-based translating services like Claude API, the system presumes consistent internet connectivity. Periodically connected devices may result in incomplete translations.
- API stability: This implementation assumes reasonable stability in the APIs of underlying translating services. Any changes to this API would require updates to the application.

It is important to understand these assumptions for setting reasonable expectations about system behaviour and identifying circumstances where additional customization would be necessary.

3.5.3 Translation quality considerations

It is very much essential to acknowledge that machine translation quality has inherent limits:

- Variations in language pairs: Translation quality varies significantly between language pairs. Hence, models with language pair available would produce better language quality.

- Context limitations: Machine translation models lack the more general contextual awareness in comparison to the human translators which can result in technically correct but contextually inappropriate translations.
- Cultural nuance: Without specific guidance, the system cannot consistently adapt material for cultural appropriateness. Elements like idioms, humour, and cultural references might not be translated precisely.
- Translation quality assessment is subjective by nature, thus absolute quality guarantees are difficult. Use case and user expectations will affect the accepted translation.

These limitations do not diminish the value provided by the application. However, in appropriate use cases and delicate settings, the possible need of human review is required.

4 The XML and JSON Translator Solution: Technical Architecture

This chapter describes the XML and JSON Translator application's technical architecture and implementation. The solution offers a specialized tool for maintaining document structure during translation, therefore addressing the challenges identified in the literature review. The system provides flexibility through multiple translation services while maintaining a consistent user experience. This section details the architectural decision, technical components, implementation details, and deployment considerations that shaped the final solution.

4.1 System Overview

The system overview provides a thorough review of the fundamental architectural decisions that shaped the XML and JSON Translator. The philosophical ideas guiding design decisions, the high-level architecture structuring the application components, and the data flow patterns allowing effective document processing are discussed in this part. Moreover, this section explored how separation of concerns, modularity and extensibility principles were used in creating a technically proficient and practically usable application. The high-level architecture discussion illustrates how the client-server model separates backend processing from frontend presentation, while the data flow analysis tracks the flow of the documents across the system from upload to translated download.

4.1.1 Architectural Philosophy

There were several architectural principles considered while designing the XML and JSON translator application. Four main principles were discussed as follow:

- Separation of concerns: Document processing service, translation services and user interface elements have clear boundaries between them.
- Modularity: Each of the components in the applications was independently developed, tested, and deployed showcasing a modular behaviour.
- Extensibility: This architecture allowed addition of more document formats and translation services to meet future demand.
- Accessibility: The application checks the accessibility principles as the user interface can accommodates for different degrees of technical expertise.

All of these principles were closely considered and followed throughout the development, resulting in the system that balances both technical capability and practical usability. The architecture followed the "clean layered architecture" as described by (Richards 2015) by incorporating clear boundaries between components and explicit interfaces for communication.

4.1.2 High-Level Architecture

The application followed the client-server architecture principles by clearly separating frontend and backend components. This distinction allowed for independent scaling and component maintenance. Figure 5 shows the high-level architecture.

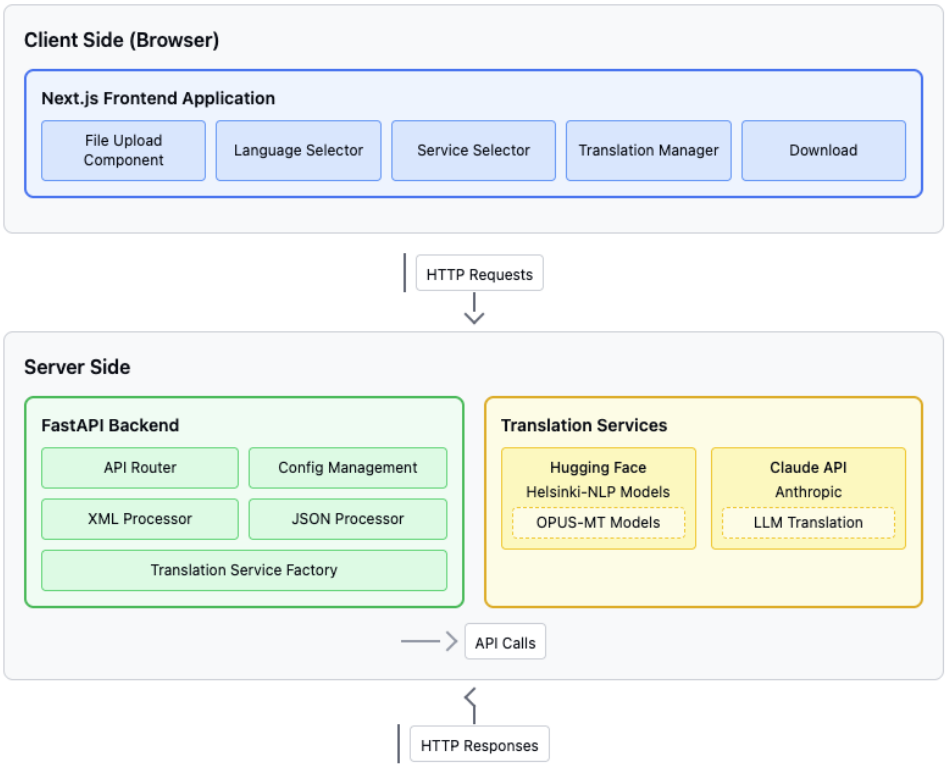


Figure 5: High-level architecture diagram showing client-server relationship with key components

This application comprises of three main components which are as follow:

- Frontend Application: A Next.js web application offering the user interface for file selection, language configuration, and results management.
- Backend API: A FastAPI service was developed in the backend exposing endpoints for document processing and translation management.
- Translation Services: This provided an integration with external services like Hugging face models and Claude API for actual translation processing.

This adaptability and maintainability of this architecture were factors in adapting it in the project. As mentioned by Fowler (Fowler, Rice, Foemmel, Mee & Stafford 2002), due to the rapidly evolving nature of the machine translation technology, it is essential to separate the business logic with the presentation concern in order to enhance the adaptability of the system.

4.1.3 Data Flow

The translation process follows a sequentially pattern for data flow as listed below:

- From frontend interface, user uploads XML or JSON documents.
- Documents were then sent to the backend API.
- Backend extracts translatable content while preserving structural elements.
- These extracted contents are sent to the selected translating service .
- Once translated, these contents are reinserted into the original structure.
- The translated document is then returned to the frontend for user download.

This whole translation flow emphasizes on maintaining the structure of the original document throughout the process, hence addressing one of the critical requirements in localization process.

4.2 Backend Implementation

The backend implementation section includes the components from server-side which powers the XML and JSON Translator application. This section provides some detail information on the selection for the technology like Python and FastAPI.

This section also explores core components discussion, document processing approach and finally translation services integration.

4.2.1 Technology Selection

The backend implementation used Python 3.10 with FastAPI as the web framework. This selection was based on several factors:

Python: Due to its readable syntax, comprehensive library ecosystem, and significant community support, Python has emerged as the predominant language for natural language processing and machine learning applications. Python was released in 1991 by Guido van Rossum and has evolved into a language that prioritizes code readability through significant whitespace and clean syntax (Python Release Python 3.10.0 | Python.org s.a.). Released in October 2021, Python 3.10 brought structural pattern matching and enhanced type annotations among other performance improvements and language tools that proved valuable for this project. This version of the Python was selected over the latest versions like 3.11 and 3.12 for its stability and broader compatibility with machine learning libraries (Why Not the Latest? Choosing the Right Python Version for Your Projects | by Anand Sahu | Medium s.a.). The extensive library of the language reduces external dependencies for core functionality implementation. Additionally, the dynamic typing system of the language allowed for rapid development cycles. Its robust built-in support for XML parsing through

the ElementTree module and JSON handling through the json module gave the XML/JSON Translator necessary basis for document processing operations.

FastAPI: FastAPI is a modern Python web API development tool. Designed by Sebastián Ramírez in 2018, FastAPI uses Python type hints and the Starlette framework to offer automatic validation, serialising tools, and documentation elements (FastAPI s.a.). Asynchronous request handling capabilities of the framework significantly outperform other python frameworks like Flask or Django when handling I/O-bound operations which is particularly crucial factor for translation services requiring external API communication (GitHub - agusmakmun/flask-django-quart-fastapi-performance-test-comparison: Python Web Frameworks Performance Comparison: Flask, Django, Quart and FastAPI s.a.). While its automatic OpenAPI documentation generating simplifies API exploration and testing, the framework's integration with Pydantic models enables strong input validation and clean error handling. These features greatly helped to reduce development time while still preserving high code quality standards.

4.2.2 Core Components

The backend implementation comprises of the core components like configuration management, API Router, Document processor, Translation Service Layer and Utility components. Figure 6 shows the layered design of backend component architecture.

Configuration Management: This component uses a settings class pattern based on Pydantic's Base Settings to provide type-safe environment variable handling and validation. This approach is considered as one of the best practices (Settings Management - Pydantic s.a.) for configuration management in production Python applications. This method helps the system to adapt to various deployment environments through environment variables rather than code modifications. The configuration system manages language model paths, CORS rules, API settings, and service credentials.

API Router: The API route implements a resource-oriented architecture with separate endpoints for XML and JSON translating tasks. The router provides a minimum interface for external consumption and clearly separates resource kinds hence following the RESTful design principles described by Richardson and Ruby (Richardson & Ruby 2008). Each route comprises of proper HTTP status code management, error handling, and input validation. The router uses FastAPI's dependency injection system to offer request context and configuration access throughout the request lifecycle.

Figure 6: Backend component architecture diagram showing module relationships

Document processor: There are separate processors with different parsing methods for handling the XML and JSON formats. The JSON processor uses recursive object traversal whereas the XML processor uses ElementTree's DOM-based parsing strategy. To preserve document integrity during translation, both processors implement careful text extraction and reinsertion logic. Their application follows the Chain of Responsibility model developed by Gamma et al. (Gangs of Four (GoF) Design Patterns | DigitalOcean s.a.), whereby document content passes several processing stages. Figure 7 illustrates the Document Processing flow.

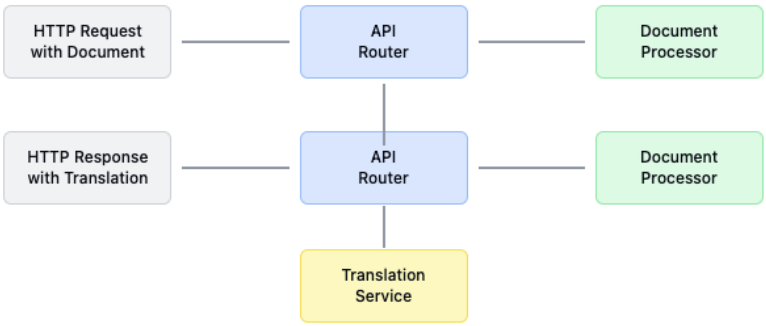


Figure 7: Document Processing Flow

Translation services layer: Underlying translating engines behind a consistent interface are abstracted in the Translation Services layer. Depending on user preferences and configured capabilities, a factory pattern offers runtime selections among two translation providers. Every concrete service implementation addresses the specific requirements of its underlying engine, including authentication, request formatting and response parsing. This abstraction layer enables easy addition of new translating providers as per the need and separates the rest of the application from changes in external service APIs.

Utility components: These components provide cross-cutting capability utilised throughout the application, including specialised logging configured for different environments, a strong error handling system that classifies errors for suitable user feedback, and data transformation utilities managing format conversions. These utilities follow reusable, cohesive helper components using Hunt and Thomas's (Thomas & Hunt 2020)design concepts.

4.2.3 Translation Services Integration

One important architectural decision for the project was using a factory pattern for translating services. Based on this, there is a central component, translation service providing common interface and rest of the application does not necessarily need to know how each service are instantiated. Depending on the user preferences, this pattern allows seamless switching between the translation services or language pair requirements. The design principle stated by Gamma et al. (Gamma, Helm, Johnson & Vlissides 1994)are followed in the pattern, which lets runtime choice of several implementations. The application implements two translation service providers:

Hugging Face models: The Hugging Face Model Hub is an online platform and repository offered by Hugging Face (Hugging Face – The AI community building the future. s.a.) where developers, researchers, and organisations can collaborate, discover, and use machine learning models. These models are especially used for natural language processing (NLP), but these models had been increasingly used for computer vision, audio, multimodal and reinforcement learning. The project uses the Helsinki-NLP OPUS-MT models from Hugging face as a primary translation engine. Following criteria were considered during the selection of these models:

- Availability: These language models have a wide range of coverage of language pair.
- Performance: Translation quality obtained from these models are reasonable specifically for technical and User Interface string translation which favoured the project needs.
- Size: These models are small enough to run locally with modest hardware resources.
- Licence: The licensing from these models is permissible for both commercial and non-commercial use.

Developed by the University of Helsinki Language Technology research group, the Helsinki-NLP models are trained on the OPUS parallel corpus, a collection of translated texts from diverse sources. These models represent the current state of the art in open-source translation technology (Thottingal 2020)and implement a transformer-based neural machine translation (NMT) architecture.

The system particularly utilised the following model configuration for each supported language:

- Finnish: Helsinki-NLP/opus-mt-en-fi
- Swedish: Helsinki-NLP / opus-mt-en-sv
- German: Helsinki-NLP/opus-mt-en-de
- French: Helsinki-NLP/opus-mt-en-fr
- Spanish: Helsinki-NLP/opus-mt-en-es
- Italian: Helsinki-NLP/opus-mt-en-it
- Dutch: Helsinki-NLP/opus-mt-en-nl
- Polish: Helsinki-NLP/opus-mt-en-pl
- Portuguese: Helsinki-NLP/opus-mt-en-pt
- Russian: Helsinki-NLP/opus-mt-en-ru

These implementations required several technical considerations including:

- Model Caching: Effective access and storage of downloaded models.
- Memory Management: limiting RAM usage for big models
- Concurrent processing: managing several translation requests without resource contention.
- Error Handling: Any failed translation attempts were directed for graceful fallbacks.

The project utilised the Hugging Face's Transformers library which offered a consistent API for model loading, input tokenisation, and inference.

Claude API Integration: In addition to the free models from Hugging face, the application also allowed the selection of the translation service using Claude API from Anthropic. Claude is a large language model (LLM) which is capable of understanding and generating human language with high contextual accuracy. The Claude integration addresses some restrictions of specialised translating models:

1. Context Understanding: This model has better comprehension of the surrounding context of the text to be translated.
2. Domain adaptation: This model shows better performance across specialised topics.
3. Instructions Following: Any translation requirement can be handled better by simply changing the prompt in the backend.

4. Quality: Complex texts translation had comparatively better quality.

The model was implemented using the Representational State Transfer (REST) API provided by Anthropic considering following technical elements:

1. Prompt engineering: Effective prompts were crafted for optimal translation quality.
2. Context Window Management: Large documents texts translation were broken down into manageable chunks for better context window management.
3. Handling responses: Translated output from the model were extracted clean without any customisation.
4. Rate Limiting: Proper management of API quotas and concurrent request limits were done.

With particular adaptations for structured document translation, The Claude API implementation followed the best practices for prompt construction from Anthropic (Meet Claude \ Anthropic s.a.), with customization done specific to the structured document translation required for the project. A snippet of prompt structure used in the project is shown below:

```
python

prompt = f"""

Translate this English text to {language_name}.

DO NOT add any introduction, explanation, or comments.

DO NOT include phrases like "Here's the translation".

DO NOT modify any XML tags or placeholders.

ONLY return the translated content, nothing else.

Text to translate:

{text}

"""
```

This prompt structure ensured that Claude only returned the translated material without adding extra explanation text that would interfere with document reconstruction.

Including Claude API as a premium translating option involved important cost considerations. As of 2024, pricing structure of Claude API is token-based. These costs vary with the model selection. The Claude 3 Sonnet model used in this implementation costing approximately $3.00 per million

input tokens and $15.00 per million output tokens(Pricing - Anthropic s.a.). Depending on language pair and document complexity, this usually results in costs between $0.08 and $0.25 per standard page of content for document translating workloads.

These costs are significantly on the expensive end than the open-source models. However, these costs remain substantially lower than professional human translation services, which average $0.10–0.30 per word or roughly $20–60 per page (How Much Do Translation Services Cost? A Comprehensive Guide s.a.; Pricing - RushTranslate s.a.). For enterprise users that need translations in regular basis, this pricing policies from Claude, place it as a reasonably affordable middle ground between free but limited-quality open-source models and high-quality but expensive human translation.

4.3 Frontend Implementation

This section discusses the client-side architecture of the XML and JSON Translator application emphasising on the user interface and interaction design. The technology stack selection and their benefits for this application are explored in this part of the report. The subsection on user interface design details how well-established usability concepts guided the development of an intuitive experience using reference to the Figma wireframe used in visualising the project during early development. The component architecture discussion maps the hierarchical organisation of UI elements and finally the state management approach sub-section reviews on the selection of this native capabilities of React framework.

4.3.1 Technology Selection

The frontend uses the combination of the technology stack like Next.js, React 18 and TypeScript 5, that reflects the current best practices in web application development.

Next.js: It is one of the powerful React framework which significantly reduces the complexity of building production-ready applications. Initially developed by Vercel in 2016, it has evolved into a complete React development tool addressing common challenges in development (Next.js Web Dev: Master this powerful React framework[Video] s.a.). Some features of the framework like enhanced server components and the App Router architecture proved particularly valuable for the translator application where user experience and productivity depend on responsive performance. The server-side rendering capabilities of the framework generate initial HTML on the server before sending it to the client. This results in faster perceived page loads compared to purely client-side rendering approaches, hence enhancing user experience by displaying the initial content faster. This framework uses file-based routing to auto-generate routes from the file structure. This

approach simplify configuration and align code with Uniform Resource Locator (URL)s for intuitive, predictable navigation.

React: It is a JavaScript library for building user interfaces using reusable, component-based architecture. React was released by Facebook in 2013. It introduced a declarative approach that simplifies UI development and improves code maintainability (Stefanov 2021). The virtual DOM implementation from React optimizes performance by updating only the necessary parts of the actual DOM. This is an important consideration when managing complex UI states translation workflows.

TypeScript: Typescript is a strongly typed programming language extended from JavaScript. Typescript was introduced by Microsoft in 2012 and it has become more popular particularly in larger codebase for its easy maintainability (Javascript & Scale s.a.). In this project development, typescript enabled clear type definitions for API responses, component props and state management. This created clear contracts between different parts of the application, reducing integration issues and simplifying maintenance.

4.3.2 User Interface Design

The process of interface design focused on providing an intuitive experience accessible to users with different technological backgrounds. To help users across the translation process, this approach needed careful consideration of usability principles, workflow patterns, and feedback systems.

A set of principles put forward from usability heuristics of Nielsen (10 Usability Heuristics for User Interface Design - NN/g s.a.) had been applied while designing this project that have become fundament to effective interface design. The project provides continuous feedback of system status throughout the application via progress indicators, validation feedback and translation status update. This help users to provide informed decisions about their next actions. The interface creates simple interaction patterns by using analogues from common web applications like drag-and-drop file uploads, dropdown selectors, and status indicators, that reduced learning requirements for the users.

Early file selection and language configuration confusion found by user testing resulted in improvements in the interaction flow. The interface was then updated that guides users through four-stages process of file selection, configuration, processing, and results review. This stepwise approach generates natural breaks for decision-making and fits with the mental models of users on how document processing workflows should proceed. Clear error prevention mechanisms were

provided by the application. These include proactive validation of file types, size limitations and configuration settings. These mechanisms helped users to avoid mistakes prior to their occurrence.

A collaborative design tool Figma (Figma: The Collaborative Interface Design Tool s.a.) was used by wireframing the initial design. Since, it was a solo project, there were not any collaboration involved during the early design decisions. The initial wireframe design established the main flow for the application from document upload through translation to result download. During this process, the main focus was given to the interaction pattern rather than visual styling. This approach allowed quick evaluation of several interface strategies including variations in file upload systems, language selection interfaces, and results presentation. Figure 8 illustrates the initial Figma design wireframe showing the user flow.

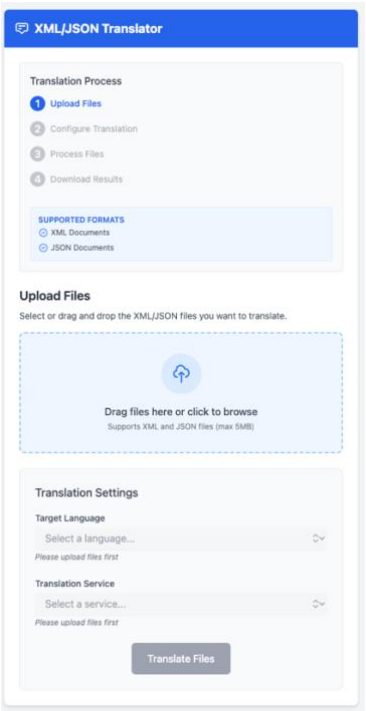


Figure 8: Figma Wireframe of the Main Translation interface.

Iterative testing of the prototype found several usability issues which guided subsequent design changes. Initially, the language selection interface showed all the options without differentiating between service providers. This caused confusion about the languages versus translation service selection. which languages were accessible with which translating service. Later designs addressed these issues employing progressive disclosure principles by presenting only necessary information at each stage.

The application used a restrained colour palette dominated by blues and neutrals was used, which created a professional appearance. The accent colours were sparingly used to highlight actions and status indicators.

4.3.3 Component Architecture

A modular component architecture was used in the frontend implementation that divides interface into independent, reusable elements. This approach allowed maintainability and enabled effective iteration through component-level testing .

The frontend architecture organises interface elements from basic building blocks to complex assemblies. This is a variation of the atomic design methodology developed by Frost (Atomic Design by Brad Frost s.a.). Some of the atomic components included in this application are buttons, input fields and status indicators. These fundamental components reduced development effort and user learning requirements by keeping constant appearance and functionality throughout the application.

The File Uploader component combined drag-and-drop capability, file validation, and status display providing a user-friendly experience. Based on the chosen service, the Language Selector loads available language options from the API and presents a simple user interface. Figure 9 shows the user-interface of the translator application.

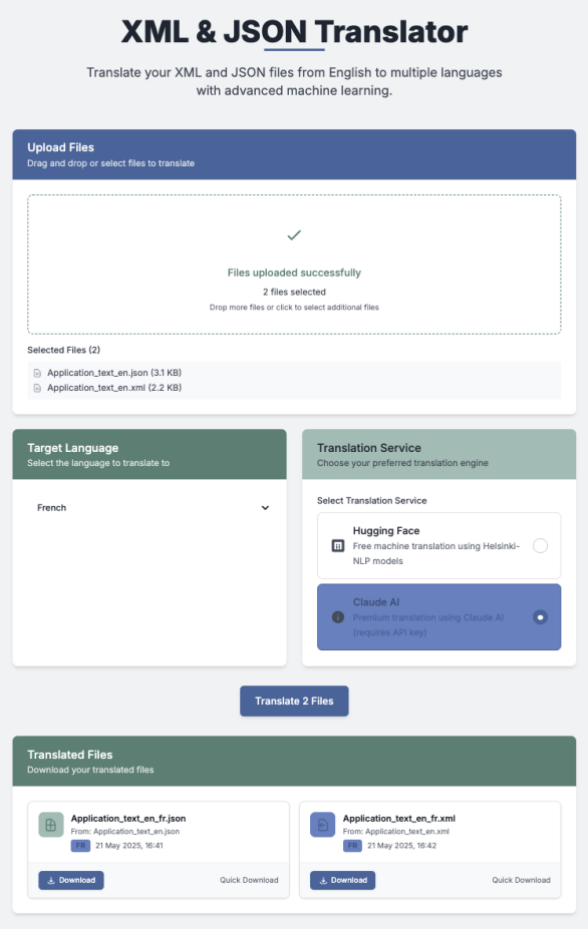


Figure 9: User Interface of the XML and JSON Translator

The User interface is connected to the backend via page components which serve as the top-level integration points. The Homepage component manages the entire translation workflow by handling API communications and co-ordinating transitions between application states.

4.3.4 State Management

The frontend application utilises the state management feature from built-in React Hooks rather than using external libraries like Redux (React useState Vs. Context API: When to Use Them s.a.). This choice was based on relatively simple state requirements of the application making the lightweight and straightforward approach of React Hooks more suitable.

The application state consists of following states:

- Upload State: Monitors files selected for translation.
- Configuration State: Captures language and service selections.
- Processing State: Tracks the progress of the translation process.
- Result State: Manages the availability of translated documents for download.

This strategy is effective for applications with straightforward data flow mechanism like the translator application hence avoiding the complexity introduced by other robust state management libraries.

4.4 Key Technical Features

This section discusses five key technical features of the XML and JSON translator tool each of which are designed to address specific challenges in the structured document translation. These features are highlighted as follow:

4.4.1 Structure preservation

This is one of the main features of the translator tool. The application utilizes the structure-aware model that maintains document integrity throughout the translation workflow, without losing critical structural elements during processing.

The system uses ElementTree library of Python to analyse the element tree of XML documents, identifying text nodes for translation while preserving the surrounding hierarchy, attributes, and special structures. For example, the namespaces, comments, and processing instructions remain intact throughout the translation process.

JSON documents have different structures compared to XML documents. For these documents, the system used a recursive traversal approach that navigates through the object hierarchy identifying potentially translatable fields based on naming patterns. In order to find translatable content, the processor used key-based heuristics to analyse field names for indicators like "text," "label," or "description," that usually indicates human-readable content.

4.4.2 Placeholder Handling

Dynamic content placeholders are one of the critical aspects of software localization, as these are the markers for variables that get replaced at runtime.

The translator application uses an automated placeholder detection and preservation system comprising C-style format specifiers ({s, %d}), named format placeholders ({ username}, {count}, template literals ({{variable}, custom delimited markers ({name})). Using pattern recognition methods, the system automatically detects these elements instead of needing human identification.

Based on the idea discussed in (Kuhn 2010) "fencing" approach, the system implementation utilized a two-phased approach. Firstly, pre-translation processing identifies placeholders with the help of regular expressions matching and then replaces them with unique markers. A mapping

between these markers and original placeholders is maintained. After translation with the help of these mapping each marker is replaced with its corresponding placeholder.

4.4.3 CDATA Handling

XML documents frequently include content like embedded (Hypertext Markup Language) HTML, JavaScript or CSS included in the CDATA sections which should not be parsed as markup. Translating these sections presents unique challenges since content in CDATA sometimes consists of translatable text as well as non-translatable code or markup. The application preserves the structural wrapping, detects these sections, separates translatable from non-translatable content, and uses specialised CDATA processing.

The CDATA handler used regular expressions, that identified and extract the content from the CDATA section. These contents are then analysed to identify potentially translatable segments while preserving embedded tags and codes. Thus, translated content is properly rewrapped in CDATA syntax to preserve document validity.

4.4.4 Batch processing

Efficient batch processing of multiple files is crucial in localization workflow. This application uses a concurrent processing system to handle several files in a single operation while providing individual status tracking and consistent results management. Localization users can upload multiple XML and JSON documents simultaneously, configure translation parameters once and initiate the process with entire batch in a single operation.

The application can manage multiple translation tasks concurrently without blocking the user interface by means of asynchronous processing approaches. Every file follows an autonomous processing path with dedicated status tracking. This means even if there is an error in translation for a single file, the system continues processing other files. Both aggregate and file-specific status updates from progress monitoring allow users to track the visibility of the overall operation and help them identify issues with particular files. This approach significantly improves efficiency for larger translation project.

4.4.5 Error Handling

Document translation involves complex processing chains including several possible failure points. The failure could occur from document parsing point to service communication to content re-insertion.

The application used a comprehensive error handling technique that prioritized elegant degradation and user recoverability over simple failure reporting. From automated retries to fallback plans to insightful user feedback, each component comprises targeted error management that detects particular failure conditions and offers suitable responses.

4.5 Implementation and Deployment

This section discusses the practical aspects of bringing the XML and JSON Translator from concept to production-ready application. Usage of docker containerization to have a consistent environment across development and deployment is discussed in the first sub-section. In the next sub-section, the development process, specifically how Trello and GitHub were combined to run the project according to the principle of Agile is detailed. The final sub-section, cloud deployment examines cloud deployment options, assessing possible host sites for the containerised application and the corresponding trade-offs. This section emphasizes the pragmatic choices that balanced the limitations of solo development with the technical ideal practices.

4.5.1 Docker Containerization

This project adopted the Docker containerisation which ensured consistent environments throughout phases of development and deployment. Docker was introduced by Solomon Hykes in 2013. It transformed software deployment with its lightweight, portable containers that packaged applications with their dependencies (MerkelDirk 2014). Conventional virtual machines virtualise whole operating systems, whereas Docker containers share the host OS kernel while keeping user spaces isolated, thus lowering the resource overhead and accelerating start-up times.

This containerising approach from Docker solves the typical "works on my machine" issue that often-beset software deployments. Docker containers ensures identical behaviour over various hosting environments by packaging the application with its exact dependency versions, runtime environment, and configuration. This translation project used docker containerization to maintain a consistency throughout development. Docker best practices were followed during development including multi-stage builds to lower image size, non-root user execution for security, and explicit dependency pinning to prevent supply chain vulnerabilities .

Docker Compose defined service relationships and dependencies in a declarative configuration file. As a result, the entire application could be started with a single command that automatically handled network configuration, volume mounting and service discovery. Additionally, this approach would also provide flexibility for several cloud platforms in production deployment by preserving the same application behaviour established during development.

4.5.2 Development workflow

The development workflow of the project used GitHub as the version control tool. As discussed, earlier sections, project management was handled using Trello. GitHub's distributed version control system offered robust branching and merging capabilities which are essential for feature-based development (Git s.a.). After each feature and bug fixes, code changes were committed to the Git version control system and merged to the main development branch. Although, this Git flow model is not ideal in software development, but for solo development project this flow provided efficiency and faster delivery. A detail overview of the project, including project installation and deployment instructions and configuration and project structure is added in the Readme file of the project which is available in the frontpage of the project in GitHub.

5 Testing and Results

This chapter presents the complete testing strategy and empirical results obtained from the XML and JSON Translator application. The application testing was conducted in a controlled development environment to assess system performance, translation quality, and user experience. The analysis consists of functional testing, performance benchmarking, comparative evaluation among translating engines, and translation quality assessment. The results and findings obtained from the testing provided valuable insight into the effectiveness of the application and identify areas for improvement in future iterations.

5.1 Testing Strategy

The testing framework used multiple evaluation strategies to ensure thorough evaluation of the capabilities of the translator application. These testing strategies were designed to address some of the requirements found during the market analysis and user research phases. The testing techniques used in the application were meant to validate both technical capability and practical utility. This method built a strong validation structure fit for the complexity of the application by combining automated testing for consistency with manual evaluation for qualitative assessment.

Functional testing part of the framework focused core application flow, like document upload, format validation, translation processing, and result delivery. The representative corpus of XML and JSON developed based on the resources from an actual project were used to stress-test edge cases. This test corpus included varying complex structure. These structures ranges from simple configuration files with minimal structure to complex enterprise application resources with nested hierarchies and mixed content types. This variation ensured that testing addressed the whole range of practical usage scenarios which are likely to be found in production environments.

Performance testing assessed document properties and system behaviour under different load scenarios. In order to set baseline performance criteria and identify possible bottlenecks, these tests measured memory consumption, concurrent request handling, and processing time. The testing framework stimulated various usage patterns of the application like single-user document processing and concurrent multi-file translating requests. Thus, obtained results offered insights on scalability properties, resource needs, network latency and external service dependencies.

Due to the subjective nature of translation evaluation, the testing strategies considered a specialized approach for quality assessment. This approach included both automated metrics and localization expert review. The testing results were given to a human evaluator to assess linguistic

quality and cultural appropriateness for a representative sample of translated material, Bilingual Evaluation Understudy (BLEU) score calculation offered quantitative comparison against some of the reference translations. These approaches addressed some inherent challenges in quality assessment and offered practical ideas for system enhancement.

5.2 Testing Results

The application showed robust performances during its workflow even when multiple files with complex structure were added for translations. Thus, reproduced translated documents contained successfully preserved document structure effectively maintaining documents validity throughout the translation process. The accuracy of the translation thus produced for some specific language was found to be around 96-97%, as some of the shorter texts have multiple translation inside the same Id. For example, when an English XML document containing a text ID "application.days" ="Fri" was translated to Finnish language, this specific content was translated multiple times giving the result text ID "application.days" = "Pe Pe Pe Pe Pe Pe Pe". Although, the translation is correct based on peer review of the translated content, this multiple translation requires more investigation. However, this was not the case for any other languages used for translation. Hence, this specific case demonstrated the requirement of human review before the translated contents were finally accepted for usage.

During the testing of various documents, it was observed that placeholder preservation was attained with 100% accuracy, hence confirming the reliability of automated detection and restoration mechanism of the application. In some cases where malformed document was used, error handling in the application provided meaningful user feedbacks keeping the system stable and functioning. This showed the successful operation of error handling mechanism in the system.

The performance of the application was affected when processing larger documents. Additionally, the performance was also affected by the content complexity and chosen translation service. Simple XML documents with less than 100 translatable elements handled in under 15 seconds using Hugging Face models while complex documents with more than 1,000 elements needed up to 2 minutes for total processing. Simple JSON documents were usually processed faster in comparison to the XML document. This is the due the simple structure of JSON documents. However, deeply nested objects in JSON documents took equivalent amount of time in processing as the XML documents. These findings established baseline performance expectations and identified document characteristics that significantly affected processing time.

5.3 Comparative performance Analysis

The translation obtained after selecting different translation services were compared during the application testing. This analysis considered various elements between these services during comparison. These comparison bases include translation accuracy, processing speed, and cost implications. A notable difference in processing speed was shown during the comparison between Hugging Face and Claude API translating services. However, translation quality comparison between these services showed no significant variations.

Processing speed analysis showed definite advantages for Hugging Face models in terms of raw throughput once models were loaded into memory. Depending on length and complexity, Helsinki-NLP models handled individual text segments in 0.5–2 seconds while Claude API requests required 3–8 seconds due to network latency and API processing time. This was also the case while batch processing multiple files.

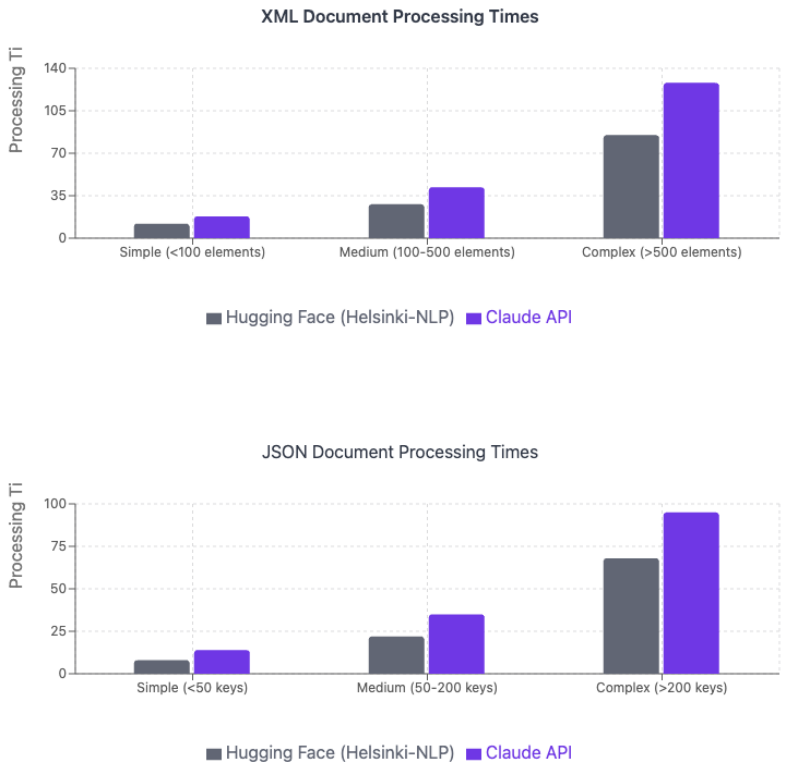


Figure 10: Comparison of Processing Times versus document elements complexities: Hugging Face Against Claude API

Figure 10 shows the comparative diagram of processing times against documents complexities for XML and JSON document formats for two different translation services. Although Helsinki NLP model showed better performance in terms of processing time, translation quality for longer and

contextual texts was better for translation from Claude due to its context awareness and natural language generation.

Cost analysis showed clear differences between the two services that greatly affected deployment decisions. Once loaded, Hugging face models incur only computational expenses that scale linearly with usage volume. However, depending on content length and complexity, Claude API pricing which is based on token consumption averaged $0.12–0.28 per document. For businesses handling less than 100 documents each month, better contextual texts translation quality from Claude could justify the additional cost. But provide its marginal cost structure following initial deployment, high-volume scenarios clearly favoured the open-source Helsinki NLP model approach.

5.4 Translation quality assessment

The translated content obtained from the application was evaluated with partial human assessment and automated metrics in order to provide comprehensive analysis of the output quality across different language pair and content type. The main quantitative indicator for evaluation was BLEU score computation, which compared system output against reference translations. Particularly for this project, the system translated content was compared against a Finnish translated content obtained from a Localization vendor. This specific translated content was analysed by a localization specialist to evaluate the accuracy hence providing a basis for partial human evaluation.

BLEU scores analysis was done across five target languages namely German, French, Finnish, Swedish and Spanish. This analysis showcased consistent trends in translation quality between services. The translated content when compared against professional human translations, BLEU scores ranging from 61.7 to 79.2 was achieved by Claude API. Claude's performance was strongest in Spanish (79.2) and German (77.3), while it scored lowest in Finnish (61.7).

Comparatively, scores from Hugging Face models were consistently low. These scores range from 43.2 to 67.1, with highest performance in Spanish (67.1) and lowest in Finnish (43.2).This analysis showed that Claude API outperformed Hugging face model by an average margin of 18.4 points, with substantial quality gap between these two translation services across all tested languages. Figure 11 illustrates the BLEU score comparison between two translation service tested 5 different languages against human translation.

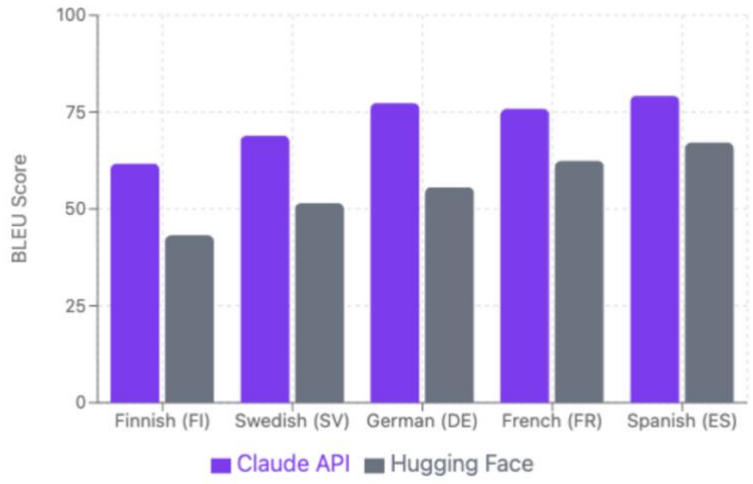


Figure 11: BLEU Score Comparison Against Human Translation

The analysis also highlighted that in some languages like Finnish and German, Claude consistently showed better contextual awareness, closely matching human translations in phrasing and technical accuracy. It also addressed structural elements, placeholders, and terminology more reliably than Hugging Face. Although both systems preserved HTML tags and placeholders reasonably well, Hugging Face occasionally changed HTML tags and placeholders, thus compromising functionality. Analysis of the translated content showed some translations from Hugging Face model, particularly for Finnish language were over-translated, had inconsistent syntax and malformed content.

These findings showed that general-purpose language models like Claude can outperform specialized specialised neural machine translation models like Hugging Face Helsinki NLP in domain specific translation tasks. The significant quality difference between the two machine translation techniques implies that depending on language pairs and quality criteria, different deployment scenarios could benefit from different translating approaches.

5.5 Content Clean-up and Optimization

Machine translation systems often produce unnecessary artifacts and inconsistencies in translated structured documents. These inconsistencies in translation could impact application functionality and translation quality. Analysis of the translated document revealed pattern of issues that required post-processing in order to ensure functional equivalence and optimal quality.

During the analysis it was observed that translations particularly from Hugging Face models contained some repeated contents, placeholder formatting errors, HTML/XML structural problems,

and inconsistent whitespace or character encoding, and placeholder formatting errors. These issues mostly appeared in Finnish and Swedish translation which required the extensive clean-up among other languages under test. There were more identified issues discovered in translation produced by Hugging Face model in comparison to output produced by Claude API. Hence, a post-processing utility was developed to perform multi-staged clean-up of the translated content. This post-processing approach identified and protected structure preservation, cleaned content from repeated elements, normalized white spacing and line breaks, restored placeholder, addressed encoding inconsistencies, and finally validated the structure integrity of the document.

This clean-up pipeline was tested against translation across five languages resulted in a clean-up of substantial number of issues specifically in the translations produced by Hugging face model.

5.6 User Experience Evaluation

The application was presented for user testing to evaluate the practical utility of the application and identify areas for improvements. User experience testing involved both usability observation and feedback collection. Participants for testing included both technical and non-technical users covering generalized user base. Since, the application was deployed to development environment only, user testing and feedback was based on the demo presented and video material shared among the participants. Observations focused on overall user satisfaction with the application interface, translation process, error recovery, and task completion efficiency. This usability testing among a varied group of possible users produced generally positive comments on the application's workflow and interface.

Some of the key usability feedbacks included are as follow:

- The file upload component of the application received particularly positive feedback. Users appreciated the drag-and-drop functionality as well as file selection functionality. Additionally, user also appreciated the clear error messages for invalid file types.
- Some users expressed desire in seeing more information about the translation models associated with every language, while selecting the language.
- The translation service selection between Hugging Face and Claude API was clear, however some users noted that a user would benefit more if detailed information about the trade-offs between these options were mentioned.
- Users liked the translation progress indication, where number of translated and remaining files were clearly displayed. Moreover, they also praised how download and results management function was smooth and had quick access to the translated content.

The main usability issue noticed was the absence of preview ability prior to download of full translations. Particularly for larger files where quality assessment could be beneficial before committing to downloading the complete document, several participants expressed an interest to see sample translations before.

Most of the users were primarily interested in the translation quality thus produced. When they were shown the comparative evaluation among the translated files for profession content from both translation services and human translation, many expressed some dissatisfaction and cited lack of contextual awareness in the machine translated content. They specifically recommended refining the logic, particularly for Claude API by improving the prompt design to enhance contextual accuracy and align more towards human-like translation quality.

However, for specific technical content translation, users were generally pleased and noted the accuracy and effectiveness of the translated content.

6 Hybrid Translation Strategy as a Business case

During the testing of the application, it was found that human translation has clear advantage over machine translation alternatives. However, if the practical trade-offs that organizations face in localization process is considered, the business case for the XML and JSON Translator remains compelling. The constant improvements in machine translation quality will help to progressively strengthen the business case. Many research trends suggested continuous improvements in neural machine translation quality (Hassan & al. 2018). In particular for technical domains with consistent terminology and structural patterns machine translation seems promising. As both machine translation technology and their own localisation requirements change, businesses should periodically reassess the quality-cost balance. With support for both open-source models and commercial APIs, the modular design of the XML and JSON Translator enables the adaption to these evolving needs without major reimplementation costs. This section discusses at the cost-benefit issues and addresses the positioning of machine translation as a complement for human translation rather than replacement.

6.1 Cost-Benefit Consideration

During the testing it was observed that human translation has better quality over machine translated content, especially in complex languages. In case of the Translator application , languages like Finnish and Swedish showed the most significant quality gaps in machine translation. However, there are several factors beyond raw translation quality that influence the business decisions. Two of the factors are cost-resources factors and Time-to-Market consideration.

Cost-resources factor: Cost in professional human translation services typically include costs based on word count, language pair complexity, and subject matter expertise. Although, Machine translation approaches comparatively have lower quality, they offer significant possibilities for cost reductions. Machine translation approaches eliminate pricing models per-word, reduce cost in project management overhead and reduce coordination costs for multi-language projects.

Time-to-Market Considerations: Machine translation provides major turnaround time advantages above possible cost savings. Based on the application testing it was shown that even complex documents could be translated in hours rather than days or weeks. This advantage can be critical for software development teams operating in agile environments with frequent software releases as per client demand.

6.2 Machine translation as complementary technology

The XML and JSON Translator application can serve as a complementary tool to human translator rather than a replacement while increasing efficiency and reducing external vendor costs. This hybrid approach aligns with observations from CSA Research (Language Services and Technology Industry Faces Revenue Decline but Remains Poised for Transformation - CSA Research > Blogs & Events > CSA in the Media > Press Releases s.a.), which underline that the successful integration of human knowledge with machine translation technologies will determine the direction of language services despite industry changes and revenue challenges. Additionally, a tiered content strategies as shown in the Table 2 can also be implemented to use the tool based on the content visibility and impact.

Table 2: Tiered Content Strategy

Content Tier	Examples	Translation Approach	Reasoning
Tier 1: High-Impact	Marketing materials, legal content, user facing strings.	Human translation only	Quality requirements outweigh cost considerations.
Tier 2: Medium Impact	Product description, help content	Machine translation + Human review	Balanced approach for quality and efficiency
Tier 3: Low-Impact	Internal docs, development strings	Machine translation only	Maximum efficiency where comprehension is primary goal.

7 Future Improvements

This section discusses over the future enhancements of the XML and JSON Translator application based on the testing results and user feedback. Mainly two critical dimension, Cloud Deployment Options, and quality enhancements to address the limitations identified in the comparative analysis are focused on this section.

7.1 Cloud Deployment Options

This docker containerized translator application architecture gives flexibility for several operational needs by supporting deployment over several cloud platforms. The application is developed based on serverless container execution approach and contains automatic scaling features, hence suitable to be deployed in cloud platforms like Google Cloud Run and various deployment options from Amazon Web Service (AWS) (Container Orchestration & Management on AWS | Amazon Web Services s.a.) through Elastic Container Service(ECS) and AWS Fargate. These services provide granular control over system orchestration and resource allocation. Additionally, these services can integrate seamlessly with broader ecosystem of AWS including load balancers, databases and monitoring services making them suitable for enterprises deployments with specific performance and compliance requirements.

The containerized architecture of the application ensures portability across these deployment options, following the "build once, run anywhere" principle that avoids vendor lock-in and allows optimization for particular operational settings. This adaptability lets the application start with basic deployment models during development and scale to more complex orchestration as needs change without requiring for fundamental architectural changes.

7.2 Quality Enhancements Priorities

Testing results showed consistent quality gap between machine a human translation especially for complex languages. The following enhancements approaches would directly address these limitations and improve overall application utility.

7.2.1 Translation Quality Improvements

In order to enhance translation quality obtained from the translator application following key improvements had been identified:

- Custom terminology management would let users provide domain-specific dictionaries by enforcing consistent use of industry-standard terms. Additionally, pre-, and post-

processing validation of the documents could be implemented in the application that will apply terminology enforcement of these dictionaries.

- In order to handle some unique linguistic structures found in some complex languages, pre-processing language specific techniques like custom tokenization and text normalization could be used. This will ensure that shortcomings demonstrated during this application testing of complex languages are addressed.
- Customized quality prediction model can be integrated in the application which will analyse the documents post translations. These models would assign confidence scores the translated output, thus highlighting only those requiring human review. Some of the test results found during the testing of these application could be used to train these models.

These improvements would directly address the quality limitations identified during the testing.

7.2.2 Usability Enhancements

User evaluation on the application usage provided several usability improvements suggestions that could significantly enhance the user experience with the translator application. Some of the improvements are as follow:

- One of the major enhancements as per user feedbacks is addition of translation preview feature in the application. This would allow user to view the translated content without downloading. These documents could be analysed in the preview section hence allowing user to assess the quality beforehand and forward the content for post processing if needed.
- A post editing interface for human reviewer could be integrated to the application. This will allow for reviewing and edit the translated content supporting hybrid human and machine translation workflow. This interface could include side-by-side comparison view, editing capabilities, version control system to track edit history and intuitive navigation for efficient post processing.
- Furthermore, possibility of having individual translated file processing in a batch processing application flow was also recommended during user evaluation. This will allow user to work independently on specific translated file without waiting, if multiple files are being translated in batch processing.

Future implementation could benefit from integrating the translator application with the current software ecosystem of the company. This integration would streamline the localization workflow by eliminating manual steps and embedding the translating tool directly into the business operations. This integration would simplify the localisation process and improve consistency and efficiency.

8 Conclusions

This thesis focused on addressing the challenges organisations face when localizing structured XML and JSON documents. These challenges include difficulties in preserving document structure, reducing vendor dependency, enabling seamless integration with technological ecosystems of the company and balancing translation quality efficiently with respect to time and cost. This project investigated how machine translation capabilities could be effectively used to translate and process structured document to develop a practical in-house solution for software localization team. As a result, a specialised XML and JSON Translator application was designed and developed.

The developed translator application successfully achieved its main technical goal of preserving structural integrity during the translating process. This was confirmed by testing the translated output and it was verified that the application demonstrated robust handling of complex elements including nested XML hierarchies, CDATA sections including HTML, and several placeholder forms. This ability of the application to preserve structures addressed a fundamental gap in some of the current solutions and offered real value even with different translation quality.

The comparative analysis of translation quality across five different languages against translation models Claude API and Hugging Face showed variation of quality based on the language complexities. This deviation in quality suggested improving the future implementation of the application to cover various language complexities while maintaining human translation for reviewing.

The results of user evaluation confirmed the practical relevance of the application highlighting some of the valuable capabilities of application were structural preservation and placeholder handling. The user feedbacks emphasized the need for improved workflow integration and preview features, laying out design guidance for further feature enhancements.

The approach of implementing containerisation in the application was proven effective to meet integration requirements and supported deployment flexibility. This architecture of the application allows organization to implement the solution within their existing infrastructure. Whether the infrastructure be on-premises or cloud-based the application would maintain consistent functionality.

The research methodology used in the project combined technical implementation with systematic quality assessment and user evaluation. This approach provided a comprehensive perspective on translation capabilities and limitations of machine translation for structured documents. This integrated approach showed that efficient machine translations workflow depends on specialized

pre-processing and post-processing tailored to document structure. This also highlighted the importance of domain-specific solution over general-purpose approaches.

In summary, the XML and JSON Translator represents a practical step towards more efficient, integrated localization workflow that align with quality, resource, and timelines constraints in software development. By preserving structural authenticity of the documents and leveraging the advances in machine translation, the application provides organizations with greater control over their localization processes reducing their reliance on external vendor and conventional barriers over internationalism. The project acknowledges current limitations in machine translations and demonstrates that targeted, domain-specific solutions can deliver immediate and meaningful value within an evolving localization technology landscape.

References

10 Usability Heuristics for User Interface Design - NN/g s.a. URL: https://www.nngroup.com/articles/ten-usability-heuristics/. Accessed: 21 May 2025.

30 Days Coding s.a. URL: https://30dayscoding.com/blog/python-web-frameworks-django-flask-fastapi. Accessed: 17 May 2025.

2025 Marketing Statistics, Trends & Data s.a. URL: https://www.hubspot.com/marketing-statistics. Accessed: 13 May 2025.

About localization files and how they're used - POEditor Blog s.a. URL: https://poeditor.com/blog/localization-files/. Accessed: 17 May 2025.

Agile Process Diagram | EdrawMax | EdrawMax Templates s.a. URL: https://www.edrawmax.com/templates/1019172/. Accessed: 18 May 2025.

Allen, J. 2006. The Unicode Standard / the Unicode Consortium. Unicode, Inc., 5, pp. 42. Accessed: 13 May 2025.

Anwar, N. & Kar, S. 2019. Review Paper on Various Software Testing Techniques. Global Journal of Computer Science and Technology, 19, C2, pp. 43–49. URL: https://computerresearch.org/index.php/computer/article/view/1873/6-Review-Paper-on-Various-Software_html. Accessed: 17 May 2025.

Atomic Design by Brad Frost s.a. URL: https://atomicdesign.bradfrost.com/. Accessed: 21 May 2025.

Beck, K. s.a. Praise for Extreme Programming Explained, Second Edition. Accessed: 18 May 2025.

Boehm, B.W. 1988. A Spiral Model of Software Development and Enhancement. Computer, 21, 5, pp. 61–72. URL: https://doi.org/10.1109/2.59. Accessed: 17 May 2025.

Cirillo, Francesco. 2018. The Pomodoro Technique . URL: https://books.google.com/books/about/The_Pomodoro_Technique.html?id=rinKDQAAQBAJ. Accessed: 18 May 2025.

Clegg, D. & Barker, R. 1994. Case Methods Fast-Track: A Rad Approach. pp. 207. Accessed: 18 May 2025.

Container Orchestration & Management on AWS | Amazon Web Services s.a. URL: https://aws.amazon.com/containers/. Accessed: 24 May 2025.

Creswell, J.W. & Clark, V.P. 2017. Designing and Conducting Mixed Methods Research | Online Resources. Sage Publications. URL: https://study.sagepub.com/creswell3e. Accessed: 17 May 2025.

CSA Research. (2020). Can't Read, Won't Buy: B2C (Summary) s.a. URL: https://insights.csa-research.com/reportaction/8057/Marketing. Accessed: 13 May 2025.

Dybå, T., Dingsoyr, T. & Moe, N.B. 2014. Agile Project Management. Software Project Management in a Changing World, 9783642550355, pp. 277–300. URL: https://doi.org/10.1007/978-3-642-55035-5_11. Accessed: 18 May 2025.

Esselink, B. 2000. A Practical Guide to Localization. 4. URL: https://doi.org/10.1075/LIWD.4. Accessed: 13 May 2025.

FastAPI s.a. URL: https://fastapi.tiangolo.com/. Accessed: 20 May 2025.

Figma: The Collaborative Interface Design Tool s.a. URL: https://www.figma.com/. Accessed: 21 May 2025.

Fowler, M., Rice, D., Foemmel, M., Mee, R. & Stafford, R. 2002. Patterns of Enterprise Application. Wesley, Addison, pp. 560. URL: https://www.oreilly.com/library/view/patterns-of-enterprise/0321127420/. Accessed: 19 May 2025.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J.M. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. pp. 416. URL: https://www.oreilly.com/library/view/design-patterns-elements/0201633612/. Accessed: 20 May 2025.

Gangs of Four (GoF) Design Patterns | DigitalOcean s.a. URL: https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns. Accessed: 20 May 2025.

Gartner Inc. (via Public) / Gartner Survey Finds 62% of Customer Service Channel Transitions are "High Effort" s.a. URL: https://www.publicnow.com/view/FC19282B6B4C2E52C4104EFFF4F5AC503E7DDDE3. Accessed: 13 May 2025.

Gengo – Professional Translation Services Within Hours s.a. URL: https://www.lionbridge.com/content-transformation-services/gengo/. Accessed: 18 May 2025.

Git s.a. URL: https://git-scm.com/book/en/v2. Accessed: 24 May 2025.

GitHub - agusmakmun/flask-django-quart-fastapi-performance-test-comparison: Python Web Frameworks Performance Comparison: Flask, Django, Quart and FastAPI s.a. URL: https://github.com/agusmakmun/flask-django-quart-fastapi-performance-test-comparison. Accessed: 20 May 2025.

Hashimoto, K., Buschiazzo, R., Bradbury, J., Marshall, T., Socher, R. & Xiong, C. 2020. A High-Quality Multilingual Dataset for Structured Documentation Translation. WMT 2019 - 4th Conference on Machine Translation, Proceedings of the Conference, 1, pp. 116–127. URL: https://doi.org/10.18653/v1/w19-5212. Accessed: 13 May 2025.

Hassan, H., Aue, A., Chen, C., Chowdhary, V., Clark, J., Federmann, C., Huang, X., Junczys-Dowmunt, M., Lewis, W., Li, M., Liu, S., Liu, T.-Y., Luo, R., Menezes, A., Qin, T., Seide, F., Tan, X., Tian, F., Wu, L., Wu, S., Xia, Y., Zhang, D., Zhang, Z. & Zhou, M. 2018. Achieving Human Parity on Automatic Chinese to English News Translation. URL: https://arxiv.org/pdf/1803.05567. Accessed: 25 May 2025.

Höppner, S., Haas, Y., Tichy, M. & Juhnke, K. 2022. Advantages and disadvantages of (dedicated) model transformation languages: A qualitative interview study. Empirical Software Engineering, 27, 6, pp. 1–71. URL: https://doi.org/10.1007/S10664-022-10194-7/TABLES/5. Accessed: 13 May 2025.

How Much Do Translation Services Cost? A Comprehensive Guide s.a. URL: https://www.pectranslation.com/post/how-much-do-translation-services-cost-a-comprehensive-guide. Accessed: 20 May 2025.

How to Understand Localization Costs: A Detailed Guide s.a. URL: https://finmodelslab.com/blogs/operating-costs/localization-services. Accessed: 14 May 2025.

Hugging Face – The AI community building the future. s.a. URL: https://huggingface.co/. Accessed: 20 May 2025.

Introducing Claude 3.5 Sonnet \ Anthropic s.a. URL: https://www.anthropic.com/news/claude-3-5-sonnet. Accessed: 17 May 2025.

Introducing Our New Subscription Pricing | Phrase s.a. URL: https://phrase.com/news/single-platform-pricing/. Accessed: 18 May 2025.

Javascript, M.Y. & Scale, A. s.a. Making Your JavaScript Applications Scale. URL: https://www.oreilly.com/library/view/programming-typescript/9781492037644/. Accessed: 21 May 2025.

Kamei, F., Pinto, G., Cartaxo, B. & Vasconcelos, A. 2017. On the benefits/limitations of agile software development: An interview study with brazilian companies. ACM International Conference Proceeding Series, Part F128635, pp. 154–159. URL: https://doi.org/10.1145/3084226.3084278;WGROUP:STRING:ACM. Accessed: 18 May 2025.

Kearns, H. & Gardiner, M. 2007. Is it time well spent? the relationship between time management behaviours, perceived effectiveness and work-related morale and distress in a university context. Higher Education Research and Development, 26, 2, pp. 235–247. URL: https://doi.org/10.1080/07294360701310839;WEBSITE:WEBSITE:TFOPB;REQUESTEDJOURNAL:JOURNAL:CHER20;PAGEGROUP:STRING:PUBLICATION. Accessed: 18 May 2025.

Key vs file-based translation management systems (TMS): challenges and benefits - Lokalise Blog s.a. URL: https://lokalise.com/blog/key-vs-file-based-translation-management-systems/. Accessed: 13 May 2025.

Kim, M., Zimmermann, T. & Nagappan, N. 2012. A field study of refactoring challenges and benefits. Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE 2012. URL: https://doi.org/10.1145/2393596.2393655;PAGE:STRING:ARTICLE/CHAPTER. Accessed: 18 May 2025.

Kuhn, R. 2010. Translating Structured Documents. Conference of the Association for Machine Translation in the Americas. URL: https://www.academia.edu/112410493/Translating_Structured_Documents. Accessed: 22 May 2025.

De La Cova, E. 2016. Translation challenges in the localization of web applications. Sendebar, 27, pp. 235–266. URL: https://doi.org/10.30827/SENDEBAR.V27I0.4942. Accessed: 13 May 2025.

Language Services and Technology Industry Faces Revenue Decline but Remains Poised for Transformation - CSA Research > Blogs & Events > CSA in the Media > Press Releases s.a.

URL: https://csa-research.com/Blogs-Events/CSA-in-the-Media/Press-Releases/Language-Services-and-Technology-Industry-Faces-Revenue-Decline-but-Remains-Poised-for-Transformation. Accessed: 25 May 2025.

Language Services Market Global | Industry Analysis, Size & Trends Report s.a. URL: https://www.mordorintelligence.com/industry-reports/language-services-market. Accessed: 13 May 2025.

Limitations of Translating Database Content with Machine Translation [2025] s.a. URL: https://www.pairaphrase.com/blog/limitations-of-translating-database-content. Accessed: 13 May 2025.

Lokalise Pricing 2025 s.a. URL: https://www.g2.com/products/lokalise/pricing. Accessed: 18 May 2025.

Manifesto for Agile Software Development s.a. URL: https://agilemanifesto.org/. Accessed: 18 May 2025.

Meet Claude \ Anthropic s.a. URL: https://www.anthropic.com/claude. Accessed: 17 May 2025.

MerkelDirk 2014. Docker. Linux Journal. URL: https://doi.org/10.5555/2600239.2600241. Accessed: 24 May 2025.

Murphy-Hill, E., Jaspan, C., Sadowski, C., Shepherd, D., Phillips, M., Winter, C., Knight, A., Smith, E. & Jorde, M. 2021. What Predicts Software Developers' Productivity? IEEE Transactions on Software Engineering, 47, 3, pp. 582–594. URL: https://doi.org/10.1109/TSE.2019.2900308. Accessed: 18 May 2025.

Next js Tutorial: A Complete Guide to Building Fast and Scalable Web Apps s.a. URL: https://akcoding.com/next-js-tutorial/next-js-tutorial/. Accessed: 17 May 2025.

Next.js Web Dev: Master this powerful React framework[Video] s.a. URL: https://www.oreilly.com/library/view/nextjs-web-dev/10000DIVC202414/. Accessed: 21 May 2025.

Paasivaara, M., Lassenius, C., Heikkilä, V.T., Dikert, K. & Engblom, C. 2013. Integrating global sites into the lean and agile transformation at Ericsson. Proceedings - IEEE 8th International Conference on Global Software Engineering, ICGSE 2013, pp. 134–143. URL: https://doi.org/10.1109/ICGSE.2013.25. Accessed: 18 May 2025.

(PDF) Personal Extreme Programming–An Agile Process for Autonomous Developers s.a. URL: https://www.researchgate.net/publication/229046039_Personal_Extreme_Programming-An_Agile_Process_for_Autonomous_Developers. Accessed: 18 May 2025.

Petrova, V. 2019. Translation Quality Assessment Tools and Processes in Relation to CAT Tools. pp. 89–97. URL: https://doi.org/10.26615/ISSN.2683-0078.2019_011. Accessed: 18 May 2025.

Pricing - Anthropic s.a. URL: https://docs.anthropic.com/en/docs/about-claude/pricing? Accessed: 20 May 2025.

Pricing • GlobalLink Strings Software Localization s.a. URL: https://www.applanga.com/pricing. Accessed: 18 May 2025.

Pricing - RushTranslate s.a. URL: https://rushtranslate.com/pricing. Accessed: 20 May 2025.

Python Release Python 3.10.0 | Python.org s.a. URL: https://www.python.org/downloads/release/python-3100/. Accessed: 19 May 2025.

React useState Vs. Context API: When to Use Them s.a. URL: https://strobecorp.com/react-context-vs-state/. Accessed: 21 May 2025.

Report: State of Localization Quality in 2023 s.a. URL: https://www.globalapptesting.com/state-of-localizatio-quality-2023-report. Accessed: 13 May 2025.

Revenue growth: Ten rules for success | McKinsey s.a. URL: https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/the-ten-rules-of-growth. Accessed: 5 June 2025.

Richards, M. 2015. Software Architecture Patterns - Understanding Common Architecture Patterns and When to Use Them. Encyclopedia of Database Systems, pp. 1601–1601. URL: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/. Accessed: 19 May 2025.

Richardson, Leonard. & Ruby, Sam. 2008. RESTful Web Services. pp. 448. URL: https://books.google.com/books/about/RESTful_Web_Services.html?id=XUaErakHsoAC. Accessed: 20 May 2025.

Settings Management - Pydantic s.a. URL: https://docs.pydantic.dev/latest/concepts/pydantic_settings/. Accessed: 20 May 2025.

Slator 2023 Language Industry Market Report - Slator s.a. URL: https://slator.com/2023-language-industry-market-report/. Accessed: 13 May 2025.

Smartling vs. Lokalise | How Do They Compare? s.a. URL: https://go.smartling.com/compare/lokalise/. Accessed: 18 May 2025.

Stefanov, S. 2021. React : up & running : building web applications. pp. 213. URL: https://www.oreilly.com/library/view/react-up/9781492051459/. Accessed: 21 May 2025.

The Scrum Guide s.a. URL: https://www.scrum.org/resources/scrum-guide. Accessed: 18 May 2025.

Thomas, David. & Hunt, Andrew. 2020. The pragmatic programmer : your journey to mastery. URL: https://www.oreilly.com/library/view/the-pragmatic-programmer/9780135956977/. Accessed: 20 May 2025.

Thottingal, S. 2020. OPUS-MT – Building open translation services for the World. URL: https://www.academia.edu/72161397/OPUS_MT_Building_open_translation_services_for_the_World. Accessed: 20 May 2025.

Tiedemann, J., Aulamo, M., Bakshandaeva, D., Boggia, M., Grönroos, S.A., Nieminen, T., Raganato, A., Scherrer, Y., Vázquez, R. & Virpioja, S. 2022. Democratizing Neural Machine Translation with OPUS-MT. Language Resources and Evaluation, 58, 2, pp. 713–755. URL: https://doi.org/10.1007/s10579-023-09704-w. Accessed: 17 May 2025.

Transifex Pricing | Plans that scale with your localization needs s.a. URL: https://www.transifex.com/pricing/. Accessed: 18 May 2025.

Translate documents | Cloud Translation | Google Cloud s.a. URL: https://cloud.google.com/translate/docs/advanced/translate-documents. Accessed: 14 May 2025.

Usability 101: Introduction to Usability - NN/g s.a. URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/. Accessed: 13 May 2025.

Vaswani, A., Brain, G., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. & Polosukhin, I. s.a. Attention Is All You Need. Accessed: 13 May 2025.

Why Not the Latest? Choosing the Right Python Version for Your Projects | by Anand Sahu | Medium s.a. URL: https://medium.com/@anand_sahu/why-not-the-latest-choosing-the-right-python-version-for-your-projects-1dd15020144d. Accessed: 19 May 2025.

XML introduction - XML: Extensible Markup Language | MDN s.a. URL: https://developer.mozilla.org/en-US/docs/Web/XML/Guides/XML_introduction. Accessed: 13 May 2025.

Appendices

Appendix 1. Code snippet for bar chart visualisation

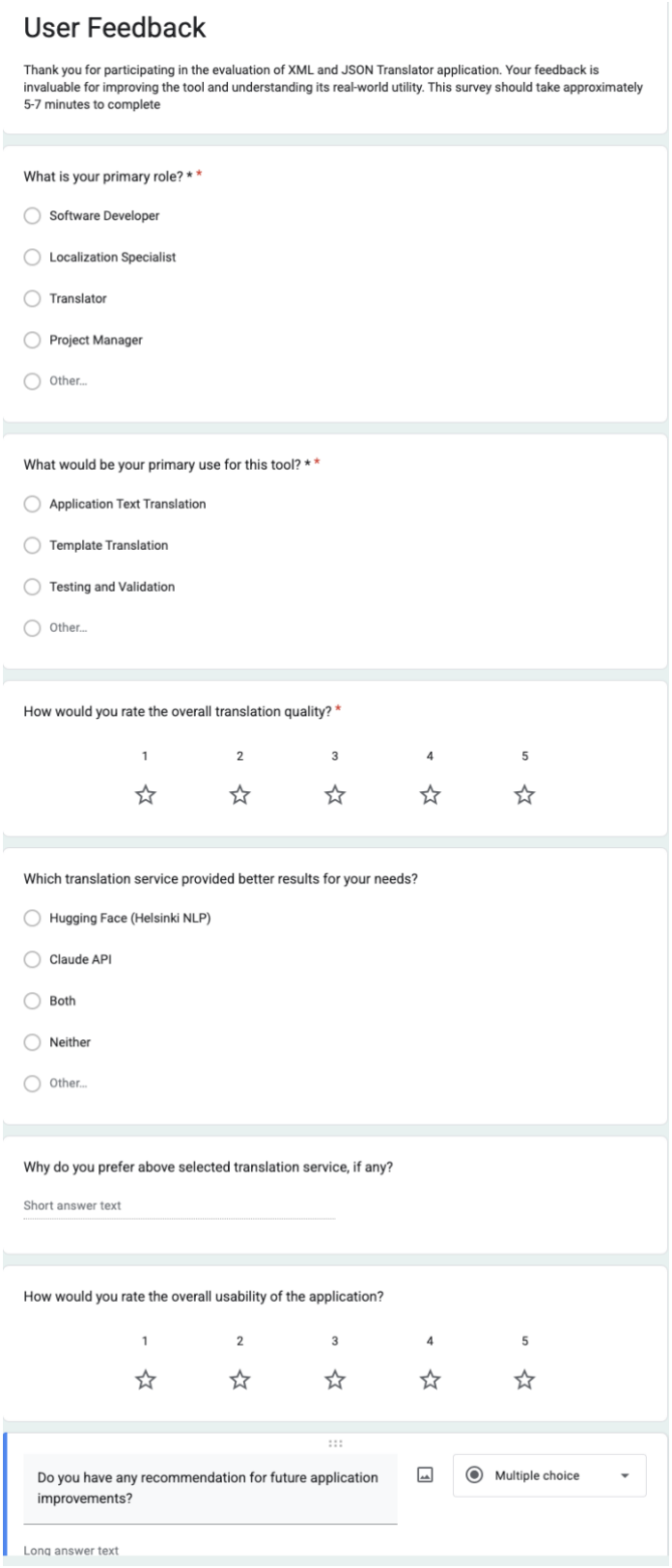
```
import matplotlib.pyplot as plt

import numpy as np

# Vendors

vendors = ["Smartling", "Phrase", "Lokalise", "Transifex", "Lionbridge", "GlobalLink"]

# Quantified data based on the best publicly available sources

pricing_tiers = [3, 3, 4, 3, 1, 1]  # Estimated from available plans

turnaround_times = [72, 60, 60, 60, 1.4, 60]  # Hours; Lionbridge = 85 min ≈ 1.4 hr

supported_languages = [150, 120, 400, 500, 70, 100]

# Set up positions

x = np.arange(len(vendors))

width = 0.25

# Plot

fig, ax = plt.subplots(figsize=(12, 6))

bar1 = ax.bar(x - width, pricing_tiers, width, label='Pricing Tiers')

bar2 = ax.bar(x, turnaround_times, width, label='Avg Turnaround Time (hrs)')

bar3 = ax.bar(x + width, supported_languages, width, label='Supported Languages')

# Labels and formatting

ax.set_ylabel('Values')

ax.set_title('Localization Service Providers Comparison')

ax.set_xticks(x)

ax.set_xticklabels(vendors, rotation=15)

ax.legend()

# Annotate values

for bars in [bar1, bar2, bar3]:

for bar in bars:

yval = bar.get_height()

ax.text(bar.get_x() + bar.get_width() / 2, yval + 2, f'{yval:.1f}', ha='center', va='bottom', fontsize=8)

plt.tight_layout()

plt.grid(True, axis='y', linestyle='--', alpha=0.6)

plt.show()
```

Appendix 2. BLEU score measurement table

Language	Claude API	Hugging Face	Difference
Finnish (FI)	61.7	43.2	+18.5
Swedish (SV)	68.9	51.5	+17.4
German (DE)	77.3	55.6	+21.7
French (FR)	75.8	62.4	+13.4
Spanish (ES)	79.2	67.1	+12.1
Average	**72.6**	**56.0**	**+16.6**

Note: BLEU scores measured against professional human translations. Higher scores indicate closer alignment with human translation quality.

Appendix 3. User Feedback forms



User Feedback

Thank you for participating in the evaluation of XML and JSON Translator application. Your feedback is invaluable for improving the tool and understanding its real-world utility. This survey should take approximately 5-7 minutes to complete

What is your primary role? * *

- Software Developer
- Localization Specialist
- Translator
- Project Manager
- Other...

What would be your primary use for this tool? * *

- Application Text Translation
- Template Translation
- Testing and Validation
- Other...

How would you rate the overall translation quality? *

1 2 3 4 5

Which translation service provided better results for your needs?

- Hugging Face (Helsinki NLP)
- Claude API
- Both
- Neither
- Other...

Why do you prefer above selected translation service, if any?

Short answer text

How would you rate the overall usability of the application?

1 2 3 4 5

Do you have any recommendation for future application improvements?

Multiple choice

Long answer text