Internationalization and Localization for Web and Mobile Applications

Peng Wang
wangkevin@cityuniversity.edu
Pinterest, Inc.

Hee Jung Sion Yoon
yoonhee@cityu.edu

Sam Chung
chungsam@cityu.edu

School of Technology & Computing
City University of Seattle

Abstract

Building web and mobile applications that quickly adapt to the language, currency, number formatting, etc., of different regions – called internationalization and localization – has become more critical for most companies since the Internet allows these applications to reach foreign customers easily. However, the high development and maintenance cost and negative performance impact are two significant problems for implementing internationalization and localization functionalities. This paper analyzes current solutions that are handling the internationalization and localization problem for web and mobile applications. The advantages and disadvantages of each approach are listed and compared. Based on the information from the analysis, a new system is designed to offer a better internationalization and localization solution with a low cost and a low-performance impact.

Keywords: Internationalization, Localization, Web Application, Mobile Application, Cloud Computing

1. INTRODUCTION

Nowadays, companies grow faster when they can ship their products globally. The software industry is also taking advantage of the Internet to deliver applications or solutions to foreign markets more than ever. However, it is hard to make an application fit into different local markets due to the language and culture differences between regions. Having a remarkable ability to handle the internationalization and localization for software becomes very crucial, which can help a company to achieve a higher customer satisfaction rate, more market share, and lower maintenance costs (Saito et al., 2017).

Internationalization in software development is a term that talks about how to develop software that can quickly adapt to other markets, i.e., other languages and cultures (Kockaert & Steurs, 2015, p. 451). Kockaert and Steurs (2015) also mentioned in their book that localization is the process of adapting a product to a local market. The localization process can include translation, date and time formatting, units converting, currency converting, and so forth.

Problem Statement

Implementing the internationalization and localization for web and mobile applications can cause a high cost during the development and maintenance process and a considerable performance impact. Therefore, managing the internationalization and localization for software can be very challenging since it will significantly increase the workload and cost due to multiple versions that may need to be created simultaneously.

Long delivering time is another problem because changing a new version may require changing every file containing text, symbols, images, videos, etc. Most importantly, adding a new feature will become more complex and time-consuming because multiple versions' software has to be maintained simultaneously. Moreover, the approach used to handle internationalization and localization may give an original system a significant performance impact due to more complexity.

Motivation
The first motivation is to find a way to allow the software to improve its user experience through internationalization and localization. Hau and Aparfcio (2014) mentioned that users always expect the software to show their languages, which can help raise productivity and significantly reduce mistakes.

The second motivation is to find a more effortless and cheaper solution for implementing and maintaining the internationalization and localization feature for web and mobile applications. According to Kidambi (2016), 60% to 80% of the total life-cycle costs for software is maintenance cost. Thus, how easy it is to maintain an application after adding the internationalization and localization solution becomes very important.

Approach
This paper evaluates how different frameworks handle the internationalization and localization problem and the non-framework way. We list the advantages and disadvantages of the existing approaches. We also compare them to find a way to improve. The ideal goal is to have a solution that can offer all the existing solutions' benefits without extra work and maintenance effort.

2. RELATED WORK

he best way to implement internationalization and localization for web and mobile applications have been discussed for a long time (Sugiura, 1986). There are many different solutions out there. Here are some popular industry solutions using front-end technologies in JavaScript:

- React applications with React-intl library (Facebook and Community)
- Angular applications (Google)
- Globalize library (jQuery Foundation)
- Android applications (Google)

React applications with React-intl library

React.js is a prevalent web application user interface library that can help developers to develop single-page web applications. It has many different libraries to help to handle internationalization and localization challenges.

The react-intl library is one of them. React-intl's (2019) official documentation can format message, date, time, number, and handle the plural issue. Developers can enable the functionality by wrapping the root component with the IntlProvider component, a higher-order component offered by the library.

A FormattedMessage component is used to tell the application to use the different messages based on the users' language setting. Another higher-order function inJectIntl is used to inject the intl object that contains format functions for the date, time, and number formatting. Using the higher-order function to wrap and inject functions makes this library very easy to use. Moreover, it also means this library will work with React library. Another downside is that the translation text files have to include the application itself, which requires republishing the application after adding a new language or updating some existing texts.

Angular applications
Angular is another popular web application framework that Google develops, used by over 1.9 million developers (2021). It also comes with its internationalization and localization solution. Angular's (2019) documentation can handle date, number, percentages, currencies, message, and plural forms of words. Moreover, Angular offers a Command Line Interface (CLI) tool to help developers generate necessary files for translators. It also can help to publish applications in multiple languages.

The following processes will be conducted after the internationalization is setup:

- Extracting localizable text for translation
- Building and serving the application with the translated message based on users' locale
- Creating multiple versions for different languages

The strength of this approach is that all of the necessary tools are included in the Angular framework, and developers can use them out of the box. It is easy to add new features with different languages since the CLI tool will extract the files automatically and allows translators to work on the text without touching any code. Moreover, this approach can be used with Angular

applications since it is an internal tool for the Angular framework.

Globalize library
Globalize is a JavaScript library that aims to offer internationalization and localization capability to web applications. According to Rosa (2016), the Globalize library leverages the official Unicode Common Locale Data Repository (CLDR) JavaScript Object Notation (JSON) data, and very easy to have the latest CLDR data (CLDR, 2019). The features of the library include:

- Number formatting
- Date formatting
- Time formatting
- Currency formatting
- Message formatting
- Plural and unit formatting

The Globalize library's (2019) official website shows that using the library is very simple. After requiring the library and loading the CLDR data, developers need to call the different formatters such as currencyFormatter, numberFormatter, dateFormatter, and so forth.

The strength of using this approach is that it will work for all of the web applications and some of the mobile applications (using JavaScript technology such as React Native or progressive web app) since it is essentially a pure JavaScript function. Another advantage is that the latest CLDR data will always be used. The most significant disadvantage is that the message module needs to load a local JSON file that contains messages in all languages, which requires republishing the application whenever changing or adding words in the file.

Android applications
Android is another popular development platform with around 3.48 million mobile apps available in the Google Play app store by the first quarter of 2021 (Statista Research Department, 2021). It also officially supports the internationalization and localization functionality of its platform. According to the Android developers' documentation (2019), Android developers can use the resource framework to separate the localized aspects from core functionality code. Android applications will switch the resources such as static data, images, videos, sounds, logos, texts, and so on based on users' languages preference. Developers can just simply put the different localized resources into other folders with the correct language naming convention. For example, the message resource for English could be placed under the res/values-en/strings.xml when the French message resource could be put under the res/values-fr/strings.xml.

The advantage of this approach is that this is a build-in tool offered by Android, which makes the workflow very clean. It also can efficiently deal with all kinds of resources besides the text, such as images, sounds, and videos. The disadvantage is that this approach works for Android since it leverages Android resource loader to switch between different resources.

Summary

After we reviewed and evaluated several different current solutions, we summarize the findings as below:

- Most of the solutions are tied to specific frameworks or platforms.
- Offering a way to extract text for translation is very important.
- Adding new languages should not require republishing.
- Updating texts should not require republishing.
- All resources such as text, image, currency, etc., should be automatically switched to the correct format based on users' preference language.
- The approach should keep developers' extra work as little as possible.
- The approach should have the ability to handle text, image, audio, video, date, time, currency, and unit formatting.

Therefore, it is good to have the ability to update CLDR data to the latest version automatically.

3. APPROACH

Our approach described in this paper for solving the internationalization and localization issue includes five parts:

1. Use plain JavaScript to fit web and mobile development with all frameworks: Using the plain JavaScript implementation can make sure the solution can be used by any frameworks such as Angular, React, Vue, and so on (Rauschmayer, 2019; Tackaberry, 2018). It can work without any framework as well (Osetskyi, 2019; Tin, 2018). The mobile applications that use JavaScript technology also work fine with this solution.
2. Separate the text content: All unrelated texts are extracted and stored in a separate file. This approach allows interpreters to work on only text files without touching the programming code.

3. Use the resource loader concept: Implementing a resource loader looks like a mechanism to allow the applications to load different images, videos, sounds, and CSS rule-based on users' languages and regions.
4. Leverage the CLDR rules: Automatically update the CLDR rules from the database to ensure the application uses the newest localization rules.
5. Use an independent cache layer to keep the resources: An in-memory cache layer is used to keep all localization-related resources such as text, images, videos, CSS rules, and so forth to reduce the application package size and allow end-users to download the necessary resources with low latency. The in-memory cache layer should also be easy to scale out with cluster mode when required.

How to integrate our system

We made our system a library and published it on a Node Package Management (NPM) system used to share code). Users can integrate and use it with the following steps:

1. Install the library into their existing system with the following comment:

```
npm install --save @kevinwang0316/i18n
```

2. Add a dictionary file with all translated text content (this is stored in the cache layer after integrating with a cache system such as Redis or Memcached):

```
// Define your dictionary for every language you want to support.
const dictionary = {
'en-US': { // Set the dictionary for the U.S. users
    login: 'login',
    confirm: 'confirm',
  },
  'es': { // Set the dictionary for Spanish users
    login: 'iniciar sesión',
    confirm: 'confirmar',
  },
  'zh-CN': { // Set the dictionary for Simplified Chinese users
    login: '登录',
    confirm: '确认',
  }
};
```

3. Import the translation text file and initialize the library (usually in the entry file):

```
// Set the dictionary to the I18n
I18n.setDictionary(dictionary);

// Optionally, you can set up a default language. If the user browser language is not found in the dictionary, this default language will be shown.
I18n.setDefaultLanguage('en-US');
```

4. Use the library in the place you need:

```
import I18n from '@kevinwang0316/i18n';

const YourComponent = () =>
<button>{I18n.get('login')}</button>;
```

How the integrated system works

The testing system has not integrated with the CLDR rules system and caching mechanism. After these two parts are done, the system works as:

1. Get the user's location setting config information from the browser.
2. One back-end call to fetch the newest translation and rule resource files comes from the cache.
3. Initialize the library with the resource files.
4. Swap the content based on the content in the resource files.

4. DATA COLLECTION

Since we design our solution as a whole system to make the internationalization and localization process easier for web and mobile applications, we collect the following data for the web application to measure the performance impact:

- Front-end CPU usage
- Page loading time
- Resources retrieving latency

All testing is conducted on the Macbook Pro 2014 version with a 2.2 GHz Quad-Core Intel Core i7 CPU and 16G memory.

Front-end CPU usage

The CPU usage was collected by using the Chrome DevTool profiling feature. The extra CPU usage that our system adds to the original system will be crucial for performance. If our system adds a considerable amount of CPU overhead, the functionalities of the original system may be

impacted a great deal. The detailed data can be viewed in Figure 1.

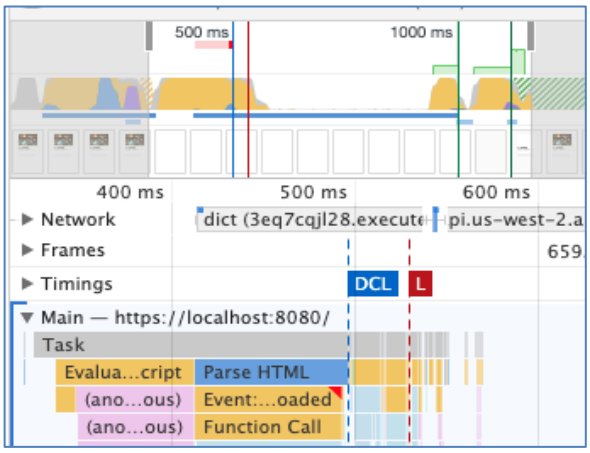

Figure 1. CPU usage

Page loading time

The page loading time can be increased significantly after using our solution since it requires loading extra resources from the Internet based on users' settings. Thus, collecting and monitoring this data is very important. It is collected by using the network module in the Chrome DevTool. The detailed data can be viewed in Figure 2.

Resources retrieving latency

Since our system can retrieve different resources such as text, video, audio, images, and so forth, the retrieving latency time should be considered vital data that has to be collected. This job can be done using AWS Cloud Watch and AWS X-Ray since our system will be integrated with AWS's services. Figures 3 and 4 show the data from AWS Cloud Watch and AWS X-Ray.

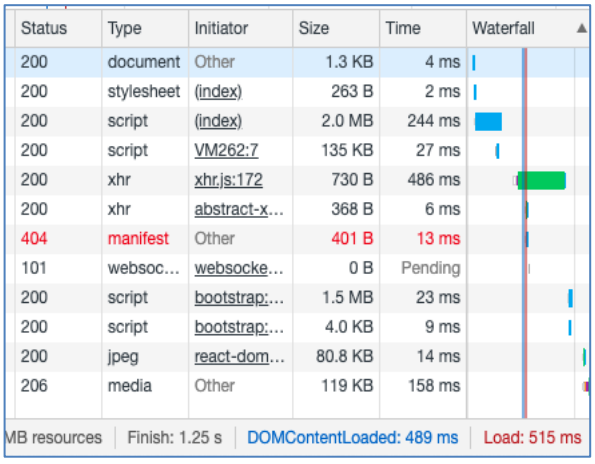

Figure 2. Page loading time

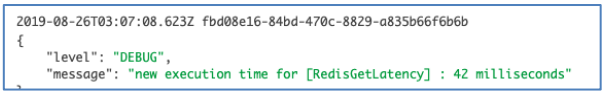

Figure 3. AWS Cloud Watch latency

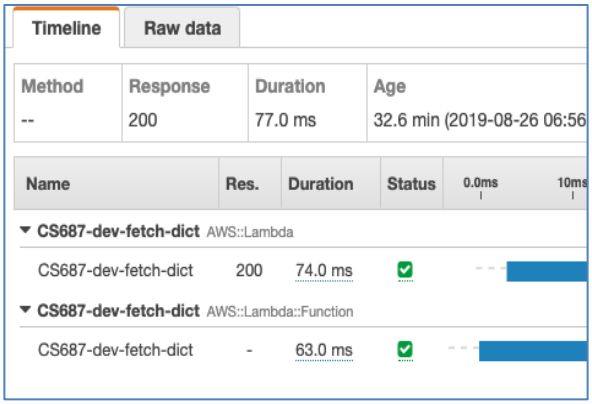

Figure 4. AWS X-Ray latency

5. DATA ANALYSIS

Three different kinds of data were collected for analysis purposes, which will be analyzed with various methods in this section.

Front-end CPU usage

The data in Figure 1 shows that our system has a shallow CPU footprint. After parsing the script, the execution phrase did not cause any high CPU usage and even finished before the HTML was parsed. Figure 1 also shows that the total script execution time is just a litter bit over 100ms.

Page loading time

Figure 2 shows that the total loading time is 515ms and the total finishing time is 1.25s. We collected the data by using the Chrome DevTool network panel. Because this test was run under the development environment that did not use a production build, the final loading time can be even lower since the production build will use multiple techniques such as minification, tree shaking, etc.

Resources retrieving latency

In the back-end code, a utility tool is written for collecting the data for a specific step. In this case, the latency of retrieving data from Redis is monitored by the utility tool and logged out to the AWS CloudWatch. Redis is an in-memory data structure store, used as a distributed, in-memory key-value database, cache, and message broker, with optional durability. Figure 3 shows that the time spends on retrieving a resource from the cache layer (Redis) is 42ms. Figure 4 shows the execution time for the whole back-end function (a warm Lambda function), 63ms.

6. FINDINGS

The finding will be shown in two parts to illustrate how our system impacts performance and whether this system is easy and cheap to use.

Performance Impact

- Low impact of CPU usage: The analysis in Section 6 shows that our system does not add any noticeable CPU impact to the original system. It means our system's impact on CPU usage is low.
- Fast page loading time: The page loading time analysis shows the whole page is loaded in 600ms. According to Google PageSpeed Insights (2019), the website will be considered fast if its First Contentful Paint (FCP) is under 1,000ms. Thus, the system does not harm the page loading time.
- Resources retrieving latency: The resource retrieving latency analysis shows latencies for resource retrieving from both the Redis and back-end function calls are very low, which will not significantly impact the original system.

Ease of Use

To use this system, we conduct the following three steps:

- Use a placeholder for all dynamic content instead of hard coding
- Create and fill out the resource template every time your system wants to add a new region support
- Add the resource template to the Redis server

Only these three steps need to be done to use the system, which is fairly to say it is straightforward to use. Additionally, adding and updating resources and other regions' support does not require any client-side or server-side code changing or redeploying, which causes the maintain cost very low.

7. CONCLUSION

Adding the internationalization and localization feature for web and mobile applications can cause a severe development and maintenance cost and a substantial negative performance impact. The system designed in this paper leverages the resource loader concept, cloud computing, and in-memory cache technology to balance developing cost, maintenance effort, and performance impact. The data collected and analyzed in the paper shows this system can help web and mobile applications handle the internationalization and localization functionality with several benefits such as a very low-performance impact in terms of CPU usage, loading page time, and resource retrieving time, a very low implementation and maintenance cost due to the ease of use.

We are not comparing the performance with other existing systems since this paper aims not to show how our system can improve the performance but to demonstrate that our system does not have a significant performance impact.

8. FUTURE WORK

There are three significant improvements to this system and could be done in future work. Firstly, make the data persistent and automatically load the data into Redis. All resource data are living in the Redis store, which is an in-memory database. More work should be done to make the data persistent and allow Redis to load the data from the data source after a crash.

Secondly, remove the back-end layer. For the demo system, the AWS ElastiCache is not used to avoid the cost. The downside of this implementation is that a Lambda function has to be used to hide the Redis credentials from the front-end code. If the AWS ElastiCache is used, the system can take advantage of the AWS assumed permission mechanism to allow the front-end code to call the Redis store directly. In other words, the back-end code can be removed completely.

Lastly, offer a tool to generate the resource template based on the existing information in the Redis. All resource information is added to the Redis store manually using a JSON format for demonstration purposes. In the future, a tool should be offered to help users to generate a resource template or event offer interface based on the current information in the Redis. It can help non-technical people such as interpreters, UI designers and handle the localization process.

9. REFERENCE

Android Developers. (2019). Localize your app. Retrieved from https://developer.android.com/guide/topics/resources/localization.html#kotlin

Angular. (2021). Github Angular repository. Retrieved from https://github.com/angular/angular

Angular. (2019). Internationalization. Retrieved from https://angular.io/guide/i18n

CLDR. (2019). Unicode Common Locale Data Repository. Retrieved from http://cldr.unicode.org

Globalize. (2019). Globalize read me. Retrieved from https://github.com/globalizejs/globalize

Google PageSpeed Insights. (2019). About PageSpeed Insights. Retrieved from https://developers.google.com/speed/docs/insights/v5/about?hl=en-US&utm_source=PSI&utm_medium=incoming-link&utm_campaign=PSI

Hau, E., & Aparício, M. (2008, September). Software internationalization and localization in web-based ERP. In Proceedings of the 26th annual ACM international conference on Design of communication (pp. 175-180).

Kidambi, P. C. (2016). Maintenance issues in software engineering. Retrieved from http://www2.latech.edu/~box/ase/tp_2003/CS532Termpaper_Kidambi_Praveen%20Chandra.doc

Kockaert, H. J., & Steurs, F. (2015). Handbook of terminology (Vol. 1). John Benjamins Publishing Company.

Margaret, R. (2017). Web application. Retrieved from https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app

Margaret, R. (2019). Mobile app. Retrieved from https://whatis.techtarget.com/definition/mobile-app

Osetskyi, V. (2019). Web application architecture. Retrieved from https://medium.com/existek/web-application-architecture-da77ea0cb520

Rauschmayer, A. (2014). Speaking JavaScript: an in-depth guide for programmers. O'Reilly Media, Inc.

React-intl. (2019). React-intl gets started. Retrieved from https://github.com/formatjs/react-intl/blob/master/docs/Getting-Started.md

Rosa, A. (2016). How to Implement Internationalization (i18n) in JavaScript. Retrieved from https://www.sitepoint.com/how-to-implement-internationalization-i18n-in-javascript

Saito, O., Boafo, Y. A., Kranjac-Berisavljevic, G., Yeboah, R. W. N., Mensah, A., Gordon, C., & Takeuchi, K. (2018). Internationalization and Localization of the Ghana Model: Lessons Learned, Opportunities for Upscaling, and Future Directions. In Strategies for Building Resilience against Climate and Ecosystem Changes in Sub-Saharan Africa (pp. 333-343). Springer, Singapore.

Statista Research Department. (2021). The number of available apps in the Apple App Store from 1st quarter 2015 to 1st quarter 2021. Retrieved from https://www.statista.com/statistics/779768/number-of-available-apps-in-the-apple-app-store-quarter/

Tackaberry, A. (2018). How to set up Internationalization in React from start to finish. Retrieved from https://www.freecodecamp.org/news/setting-up-internationalization-in-react-from-start-to-finish-6cb94a7af725

Tin, F. (2018). Automated software internationalization and localization. Retrieved from http://www.freepatentsonline.com/10078504.html