lokalise

How to solve biggest localization issues for developers

Discover which 9 localization issues developers face and what you can do to solve them. Make your developers happy, optimize your costs, increase productivity, and get to market faster.

Table of contents

Introduction

Implementing a proper localization workflow can be a rather complex task. There are many things to consider and keep in mind. When it comes to localizing software or any other digital assets, in addition to translating text, you also have to account for context and usability. This means adjusting date formats, layouts, meeting legal requirements, and more.

Traditionally, software localization has involved a lot of developer hours. The lack of dev time is the biggest bottleneck when it comes to releasing multilingual software on time.

In this guide, we will discuss some of the most common localization issues faced by developers and how to solve them with the help of solid localization software. You will learn how to automate the process, thus minimizing dev involvement across the entire localization workflow.

01

Wasting time on manual processes

The number one pain for any developer is performing routine manual tasks. Using Excel, Word, and the like for translation purposes, and then trying to copy/paste these translations into the software's code, is a time consuming and error prone process. An alternative scenario we have seen is when a developer is asked to send a source file to translators. In turn, the linguists make edits directly in this file and send it back. This, however, leads to all kinds of problems, such as:

- It is hard to resolve conflicts and make sure that the same file was not edited by multiple people
- Certain file formats can be quite complex and you cannot expect translators to understand each of them
- It is easy to break the file by adding an extra space or a quotation mark
- If the file is broken, developers will need to spend additional time on finding the problem
- There's no easy way to make sure that all messages are translated into all languages

All of the above issues involve a lot of manual work and this results in wasted dev hours. Imagine five files (one per language), and each file contains five hundred entries. How would you match them and see which keys are missing?

Takeaway: Software developers need proper tools to automate processes and stop sending on translation files manually.

02

Synchronising translations between TMS and code repository

For teams using a translation management system (TMS), devs need to make sure that translation files are synchronized properly between the TMS and the code repository hosted on platforms like GitHub or Bitbucket.

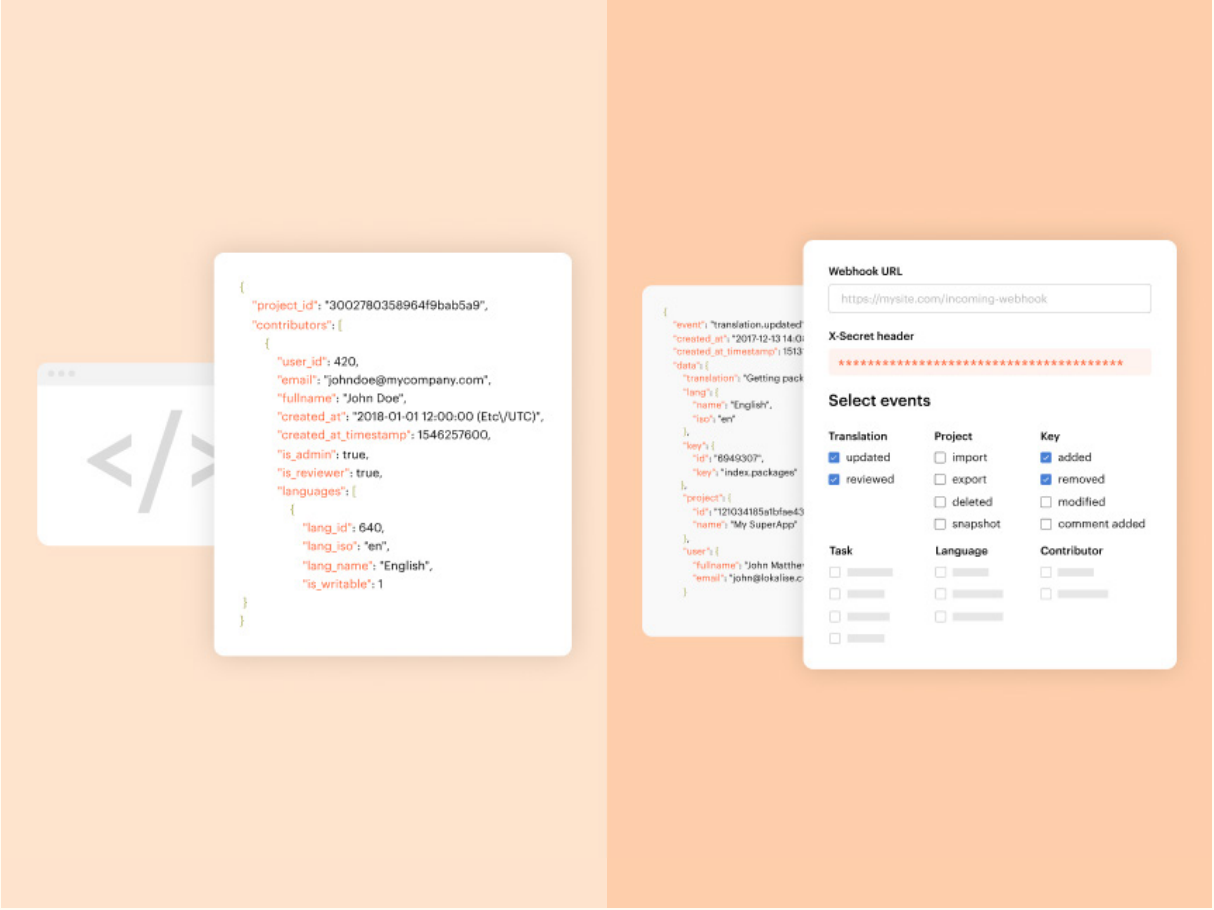
How do you achieve this? Do you export the translations from the code repo to the TMS manually? Do you create pull requests each time translators perform updates? You can't also expect translators to perform these tasks themselves. Perhaps they aren't the most tech-savvy people, or they don't have experience in tasks outside their usual role, after all. However, carrying out these tasks manually is quite a tedious process. Is there any way to alleviate this localization issue? The answer is yes!

Lokalise provides integrations with popular code hosting services: GitHub, Bitbucket, and GitLab. By employing them, developers can easily import and export translations with just a couple of clicks. This way, tiring back-and-forths are eliminated, as is idle waiting time.

Takeaway: Even when localization teams use translation management systems, the issue with passing around translation files may still persist without ever establishing proper synchronization with a code repository.

03

Downloading new translations and monitoring for changes



The next localization issue developers often experience is the need to download translation files to your software on a regular basis. Apart from that, some developers may need to monitor changes made by translators in the source files: for example, to check which translations were added or updated.

To overcome these issues, Lokalise provides two powerful tools: API and webhooks. Let's start by discussing the API.

Lokalise API

API means “application programming interface”. It allows third-party applications to communicate with the given service and manage it by sending properly formed HTTP requests.

Lokalise provides a feature-rich API that enables you to perform all major tasks, including file import and export, translation updates, contributor management, project creation, and many more. There are client libraries for major programming languages such as PHP, Go, JavaScript, Ruby, and Python.

Example: downloading translations

```
require ‘ruby-lokalise-api’
require ‘open-uri’
require ‘zip’
require ‘yaml’
client = Lokalise.client ‘your_token_here’ # 1
opts = { # 2
  format: ‘yaml’,
  placeholder_format: :icu,
  yaml_include_root: true,
  original_filenames: true,
  directory_prefix: ‘’,
  indentation: ‘2sp’
}
uri = client.download_files(‘your_project_id’, opts)[‘bundle_url’] # 3
Zip::File.open_buffer(URI.open(uri)) do |zip| # 4
  zip.each do |entry|
    next unless /\.ya?ml/.match?(entry.name) # 5
    filename = entry.name.include?(‘/’) ? entry.name.split(‘/’)[1] : entry.
name # 6
    data = YAML.safe_load entry.get_input_stream.read # 7
    File.open(“locales/#{filename}”, ‘w+:UTF-8’) do |f| # 8
      f.write(data.to_yaml) # 9
    end
  end
end
```

So, this program works in the following way:

1. We create an API client using the ruby-lokalise-api gem. Then:
2. Generate a new hash with download options.
3. Download the files and extract the bundle_url from the response. This URL points to the ZIP archive with your translation files.
4. Unzip the archive and process its contents.
5. Check the filename and make sure it has a .yaml or .yml extension.
6. The filename may contain slashes meaning that the file is stored under a subdirectory. In this case take only the filename, without the subdirectory part.
7. Read the file contents and process it as YAML.
8. Create or open an existing file under the locales directory. The filename was already generated in step #6.
9. Save translations to the file.

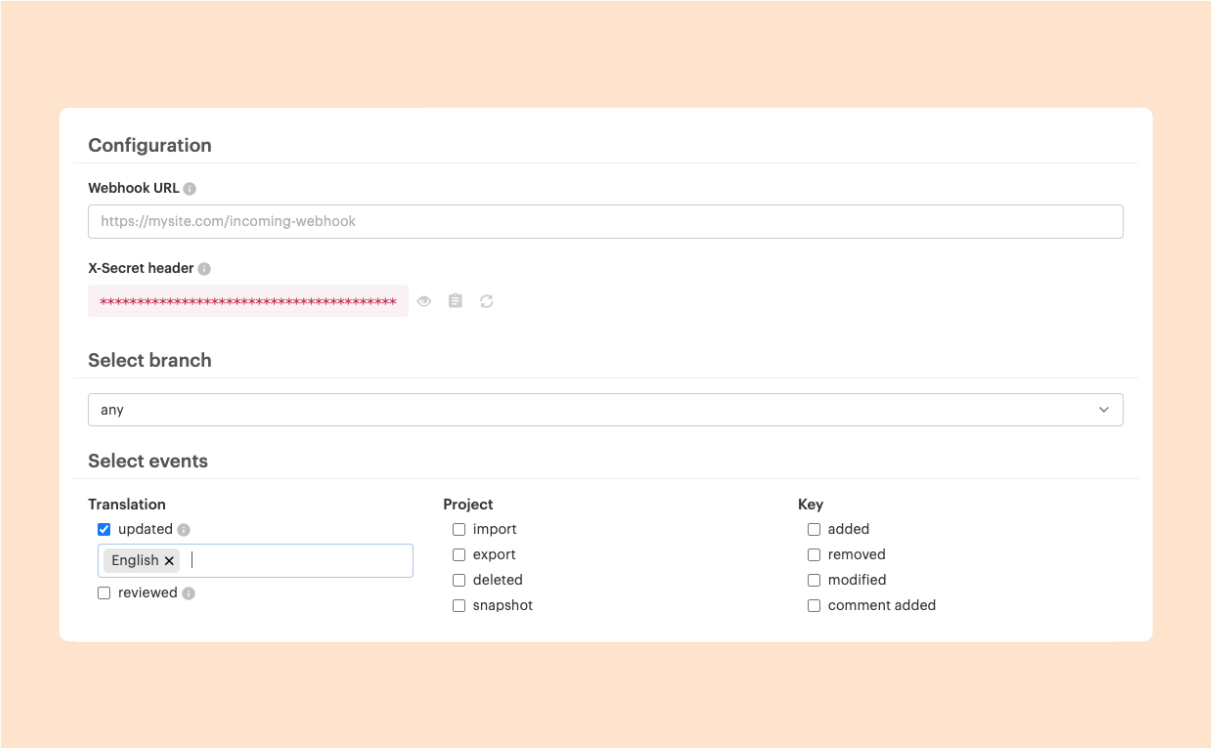
You can run the above script on a regular basis to download new translations to your application.

Webhooks

A webhook is a mechanism that sends an HTTP request to the specified URL once a given action has taken place. With the help of Lokalise webhooks, you may notify your application about certain events.

Let's look at an example set up at Lokalise.

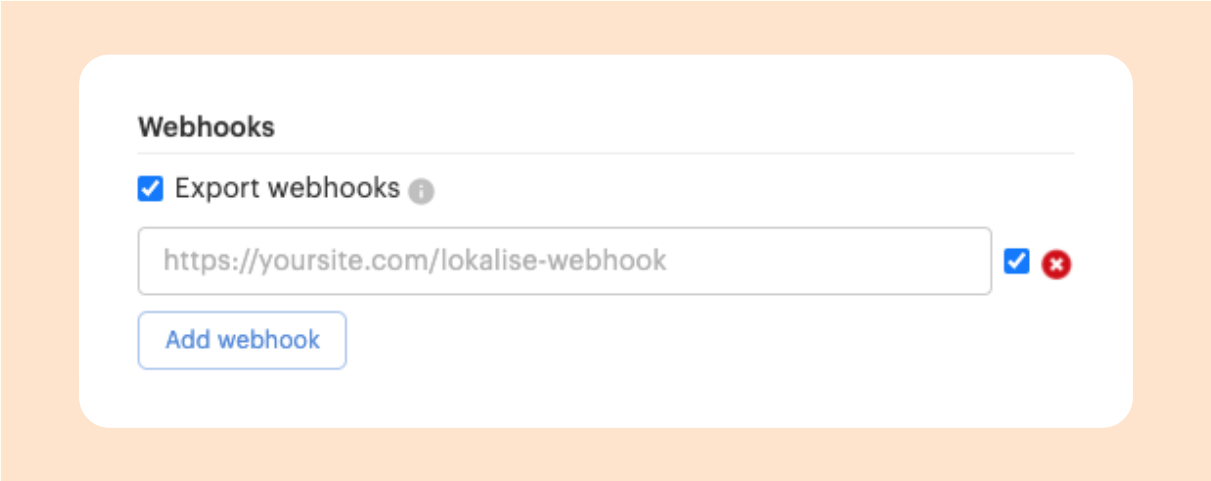
To add a new webhook, open your Lokalise project, click the "More" button and proceed to Integrations. Find "Webhook" and click "Connect". Next, enter the Webhook URL to which you want to send notifications. Take a note of the X-Secret header which can be used in your application to make sure that the request comes from a trusted source. Finally, choose one or more events to send notifications about.



Now, when a chosen event takes place the corresponding notification in the form of an HTTP POST request will be automatically sent to the provided URL. Here is an example of a “translation updated” notification:

```
{
  ‘event’ => ‘project.translation.updated’,
  ‘translation’ => { ‘id’ => 406_293_058, ‘value’ => ‘Everything about
us’, ‘previous_value’ => ‘About us’ },
  ‘language’ => { ‘id’ => 640, ‘iso’ => ‘en’, ‘name’ => ‘English’ },
  ‘key’ => { ‘id’ => 56_711_483, ‘name’ => ‘menu::about’,
    ‘filenames’ => { ‘ios’ => nil, ‘android’ => nil, ‘web’ => ‘%LANG_ISO%.
json’, ‘other’ => nil } },
  ‘project’ => { ‘id’ => ‘4578f32954cb9.016’, ‘name’ => ‘My Project’ },
  ‘user’ => { ‘full_name’ => ‘John Doe’, ‘email’ => ‘john@doe.com’ },
  ‘created_at’ => ‘2020-09-28 16:17:38’, ‘created_at_timestamp’ =>
1_601_302_658
}
```

Also note that you can send a notification using webhooks when exporting translation files. For this, simply navigate to the Download page, enable the ‘Export webhooks’ option and provide your webhook URL:



In this case, the notification will contain a path to the ZIP archive with your translations.

Takeaway: You can build your own integration and customize notifications using a powerful API and the Webhooks functionality.

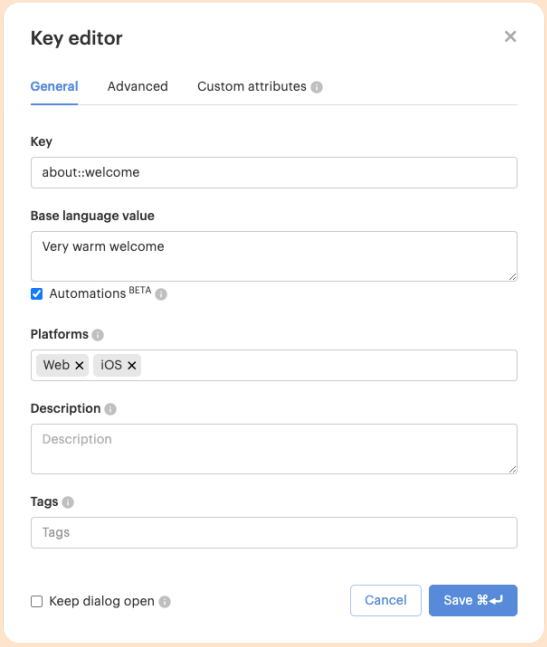
04

Having proper translation file formats

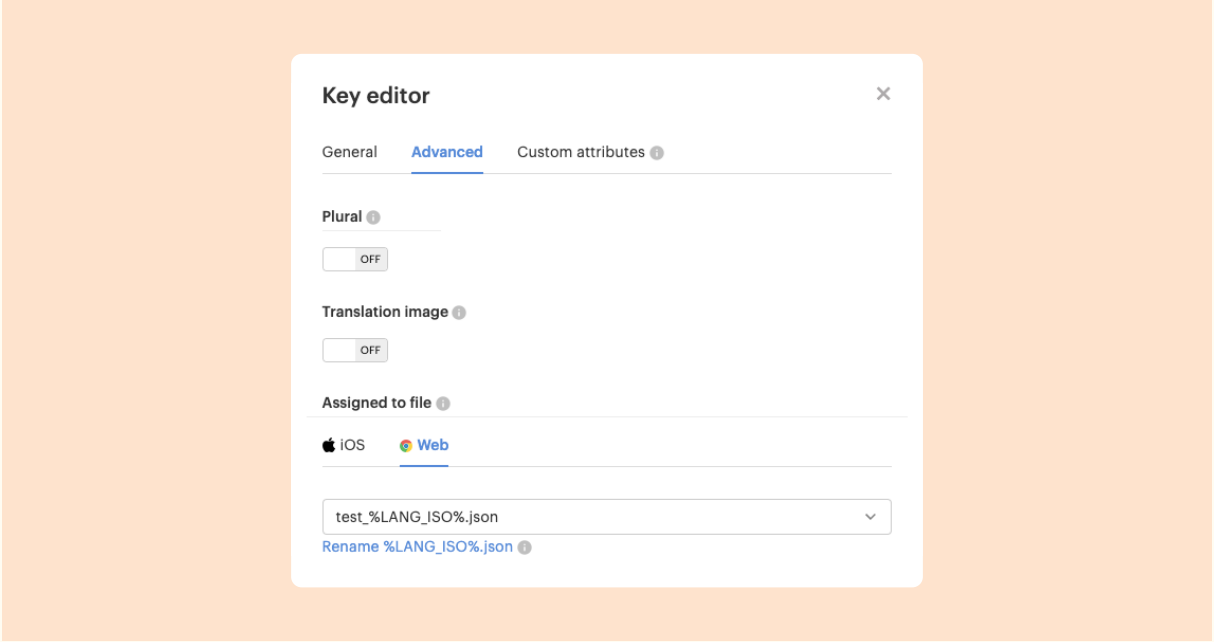
Translation file formats vary widely depending on the platform and technology. There are formats like JSON, XLIFF, YAML and many others. You need an easy way to download translations in different formats for different platforms (say, websites or mobile apps).

Assigning keys to platforms and files

First of all, with Lokalise you can assign translation keys to one or more platforms, namely Web, Android, iOS, and Other. This way, you can control which keys to download for specific formats. For example, if a key is assigned to Web and iOS platforms, it will be exported when choosing JSON and Apple XLIFF formats.



You can assign translation keys to separate files. Moreover, if the key belongs to multiple platforms, file names can be customized separately. Using this feature, you may control which key should be placed into which file during export.



Download options

The most important options include:

- **Format.** Here you may choose one of the file formats to use. These formats are separated by platform (Web, Android, iOS, Other). Therefore, if a key only has a Web platform selected, it won't be exported for Android (though you may export all keys regardless of their platform).

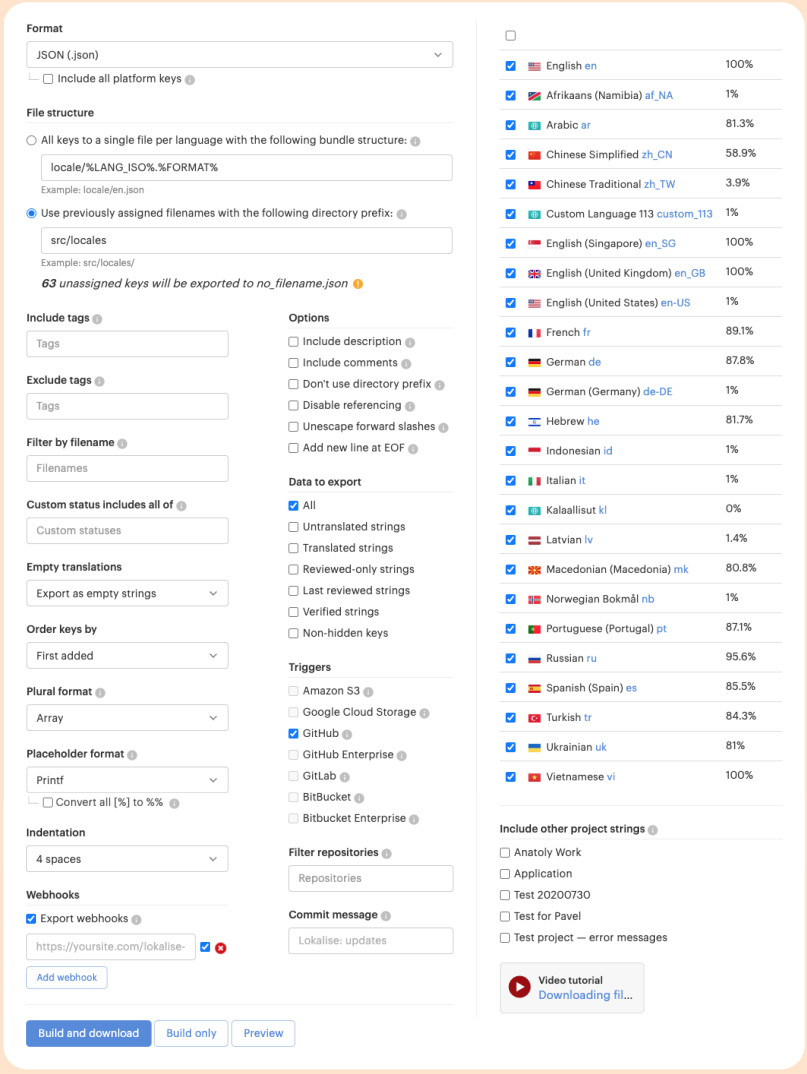
- **File structure.** Pick how your translations are organized: place all translations in a single file per language or utilize previously assigned filenames. Here you may also provide directory prefixes which is convenient when translation files for different languages should be placed into different folders.

- **Filter by filename.** Export only the keys that were assigned to the specified files.

- **Plural and placeholder format.** Select the format that should be provided for your technology. The choices will vary based on the chosen format.

- **Data to export.** Decide what keys to download. For example, you may omit all untranslated strings or download only the strings that were verified.

- **Languages.** Choose one or more languages to download translations for.



Takeaway: Assigning keys to various platforms within the localization tool simplifies the process of downloading keys in different formats and for different platforms when needed.

05

Finding duplicate translations

When a team of developers works on the same project, they may end up producing duplicated translations. For example, someone may create a new webpage with a license agreement and add an "OK" button there. To provide translation, one of the developers will also add a key named "license_ok" with an "OK" value. However, another person may also introduce a different translation key, "accept", with the same "OK" value. This means that this value has to be translated twice which is not ideal. Moreover, there can be multiple translation files for different platforms that have the same translation values but different key names. This may result in a total mess really quickly.

The solution to this? The duplicate finder feature in Lokalise.

Using this finder, developers are able to detect all the duplicated translations and decide what to do about them:

- Delete completely
- Hide for all non-admins (perhaps, to decide later)
- Merge all duplicate translations into a single key
- Create a so-called "parent" translation key and link all duplicates to it (when the parent translation is updated, all the linked keys are updated as well)

Takeaway: The duplicate finder feature helps you avoid translating duplicate strings over and over.

06

Version control

Developers love version control systems like Git. They allow you to have a base version of the project and work on new features in other branches without affecting the master. Also, you can have different developers working on separate features in parallel, which is crucial for larger teams. Can this approach be applied to a localization workflow? But of course! In Lokalise you can create as many branches inside the project as needed and switch between them in just two clicks.

When a feature is finalised, the branch can be merged with the base project version (called master or main). After merging, the master version contains both previous changes and the changes taken from the other branch. Any conflicts found during the merging process can be easily tracked and resolved. For example, if the same line was modified in both branches, you will be able to decide what changes to keep.

By using this feature, different translators can work on different areas of projects with ease.

Takeaway: The branching (version control) feature allows you to work on a new version of your content, while simultaneously supporting previous versions.

07

Proper use of placeholders and plural keys

Universal placeholders

Another common localization issue is the proper use of placeholders. As you probably know, placeholders allow you to insert dynamic values right into the translations (for instance, to greet a currently logged in user). The thing is, different formats require different placeholders. If the same translations are utilized for multiple platforms, that may become an issue as developers will have to somehow adapt placeholders manually for each case.

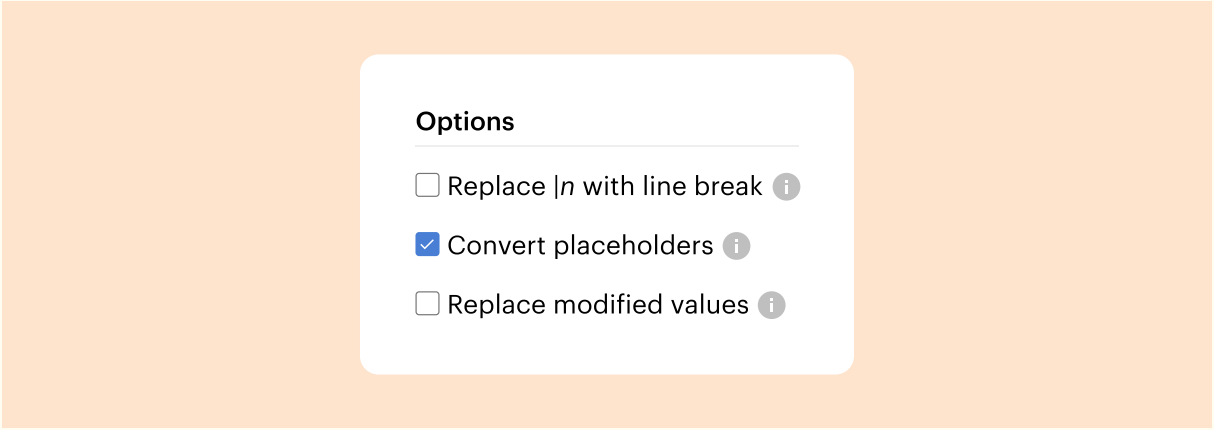
This problem can be solved with universal placeholders.

Suppose you are uploading a YAML translation file as below:

```
en:
  welcome: “Hello, %{username}”
```

Such files are used in Ruby on Rails applications. However, if you also need to export this translation for Android, the `%{username}` placeholder should be replaced with `%1$s`. Of course, you don’t want to make this change manually, right?

With Lokalise, before importing your initial translation file, you can enable the ‘Convert placeholders’ option.



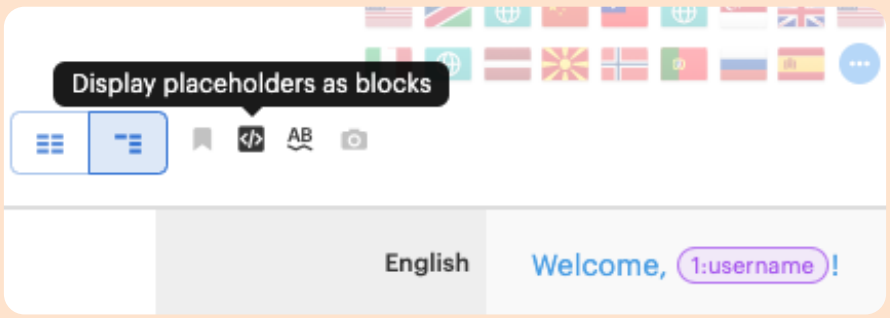
When this option is checked, all your platform-specific placeholders will be converted to universal ones. For example, `%{username}` becomes `%1$s:username`. Yet, the real magic happens when you download your translations again for the specific platforms. During export the universal placeholders will be replaced with the platform-specific ones automatically! This means that the same translations can be utilized on multiple platforms without any more worrying about the placeholders.

In the export options, you can always choose how to format placeholders (these options will vary depending on the chosen file format):

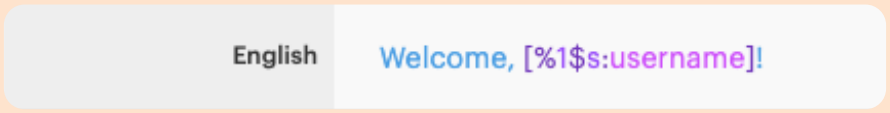
Showing placeholders as blocks

Another problem with placeholders is that some translators tend to edit them (or remove part of them by mistake), thinking that they have to be translated. This is actually not the case: they need to be left intact. Therefore, you need an easy way to show that the placeholder is a special construct and should not be modified in any way.

Here's how Lokalise provides a simple fix for this issue. Placeholders are displayed as blocks in the graphical editor, enabling translators to visually differentiate regular text and "system" elements which have to be left as is.



Even if this option is not enabled, placeholders will still have a special highlighting, for instance:



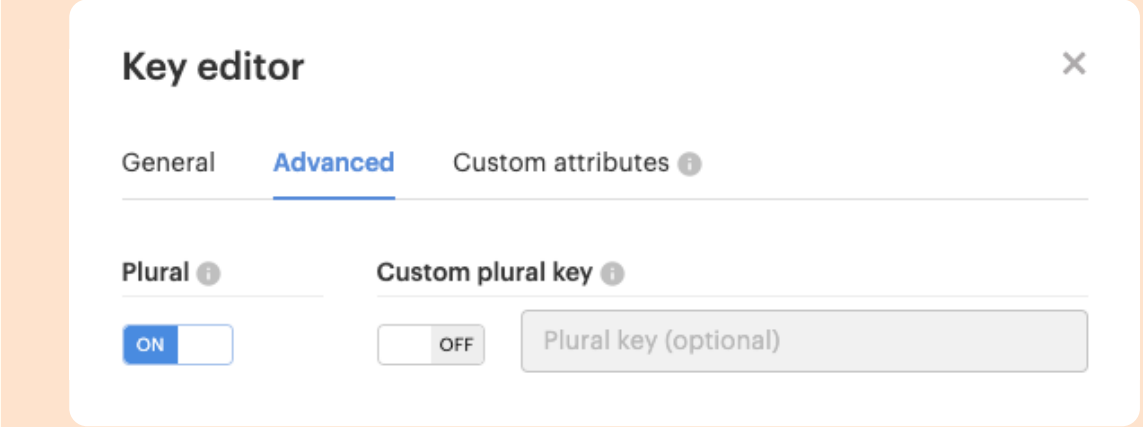
Working with plural keys

Placeholders are not the only thing that is platform-specific: unfortunately, plural keys have very different formatting rules as well. Moreover, different languages have different plural forms. For example, in English, there are two plural forms, whereas in Russian there are four. No one can possibly know all these specifics for every language on Earth. This is why you need software to help out.

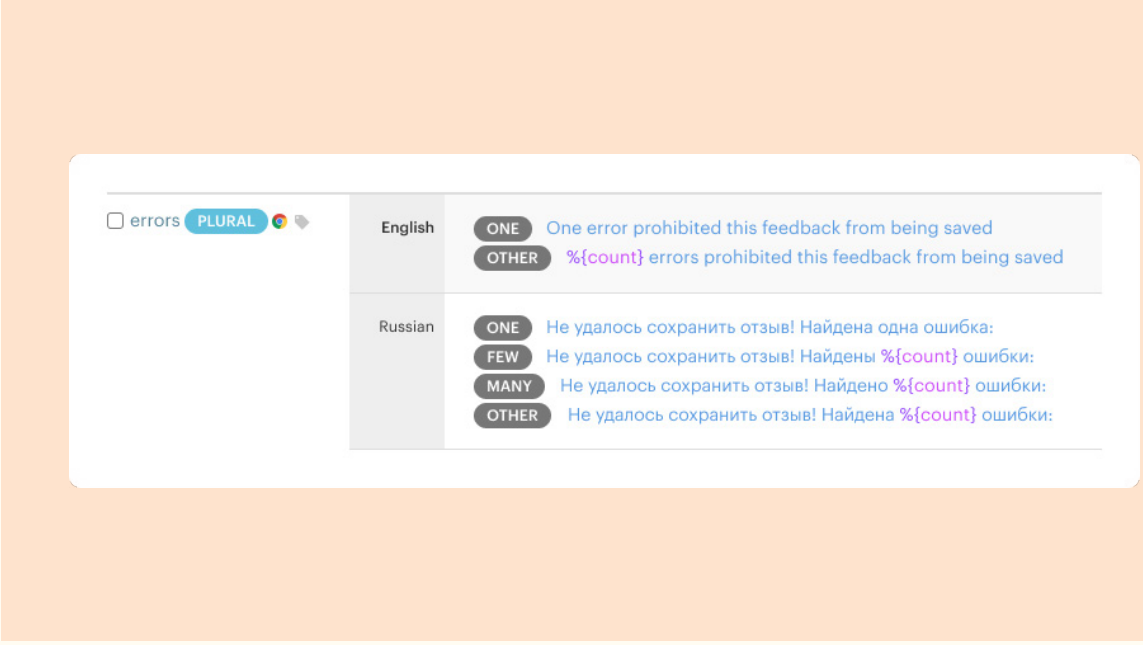
Lokalise automatically exports plural keys in the proper formats, and has built-in support for virtually any language on Earth, including their plural forms.

How it works

To make a key plural, open its settings and toggle "Plural" to On:



The plural forms will be provided automatically for each added project language:



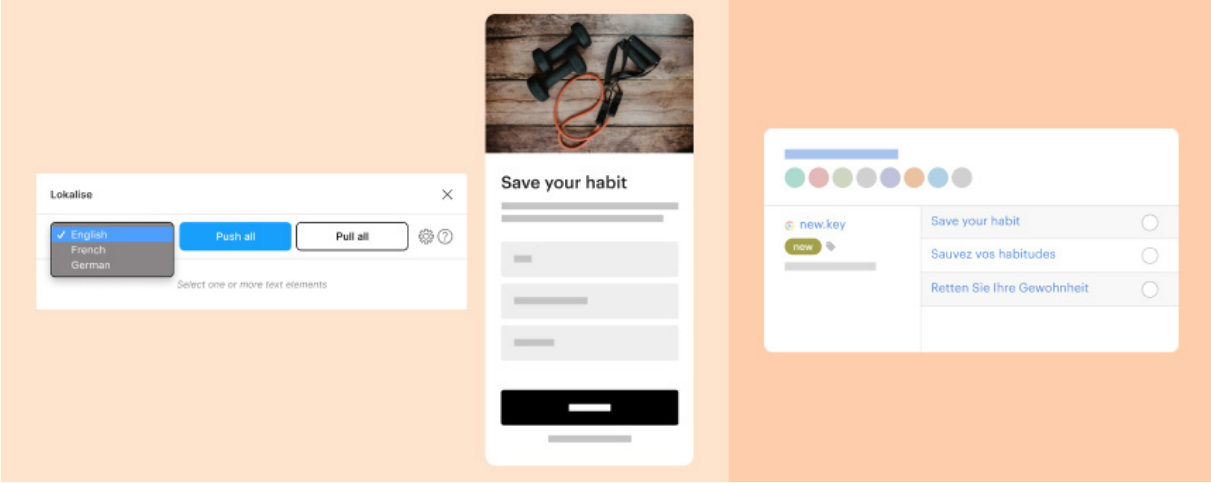
If you need to change the default plural forms, you can also do so in the Language settings.

Similar to placeholders, during export, you can choose the plural format to apply (these options will vary depending on the chosen file format):

Takeaway: Placeholders and plural keys are platform specific. Having an option to select the appropriate formats during export eliminates the need for developers to adapt them manually.

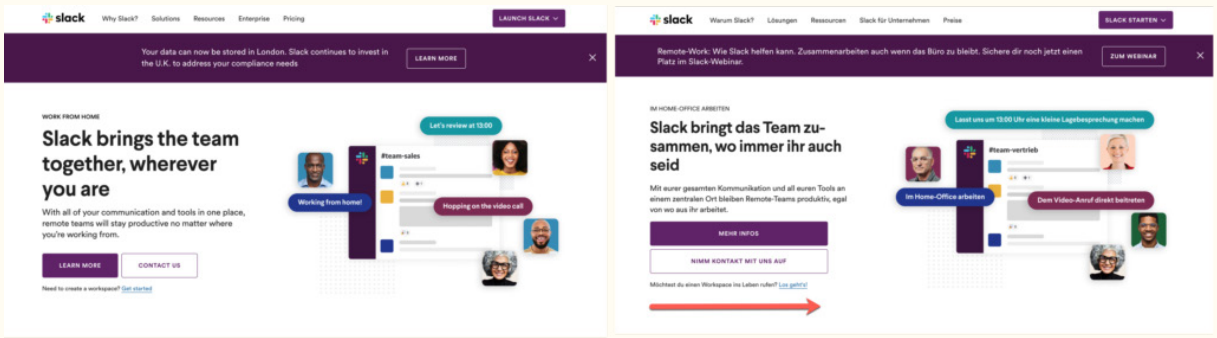
08

Ensuring translations fit the design



This localization issue affects both developers and designers. Both groups need to work together to ensure that translations for different languages fit properly within the design. This is very important because certain phrases in some languages may become significantly longer or shorter, thus breaking the layout.

For example, translating from the English language into German can result in a 20-35% text expansion. Translating from English to Swedish can result in a 20-35% text contraction.



slack.com vs slack.com/intl/de-de/

Plus, if you want to include Korean, Japanese, Chinese, and other non-latin script languages, this can also result in vertical text expansion.

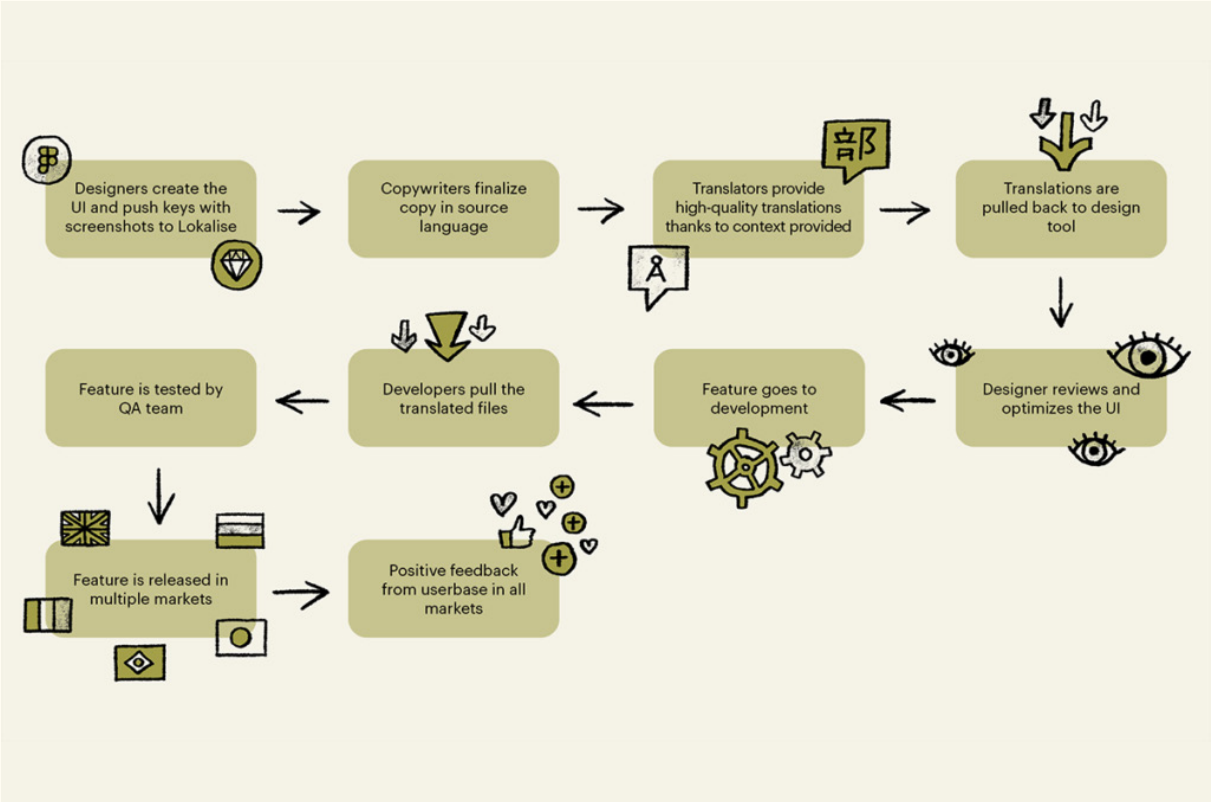
slack.com/intl/ja-jp/

If you work with translations only after the development phase, you'll encounter the following challenge, which could have been avoided or at least significantly reduced:

- Whatever changes or fixes are needed, they will have to go through the engineering team who will either fix bugs or break new texts into keys, and sync them with existing keys.

Start Translating in the Design Stage

With Lokalise, you can add designers (and copywriters) into the mix early on. The Lokalise integrations with Figma and Adobe XD make it possible to start translating in the design stage.



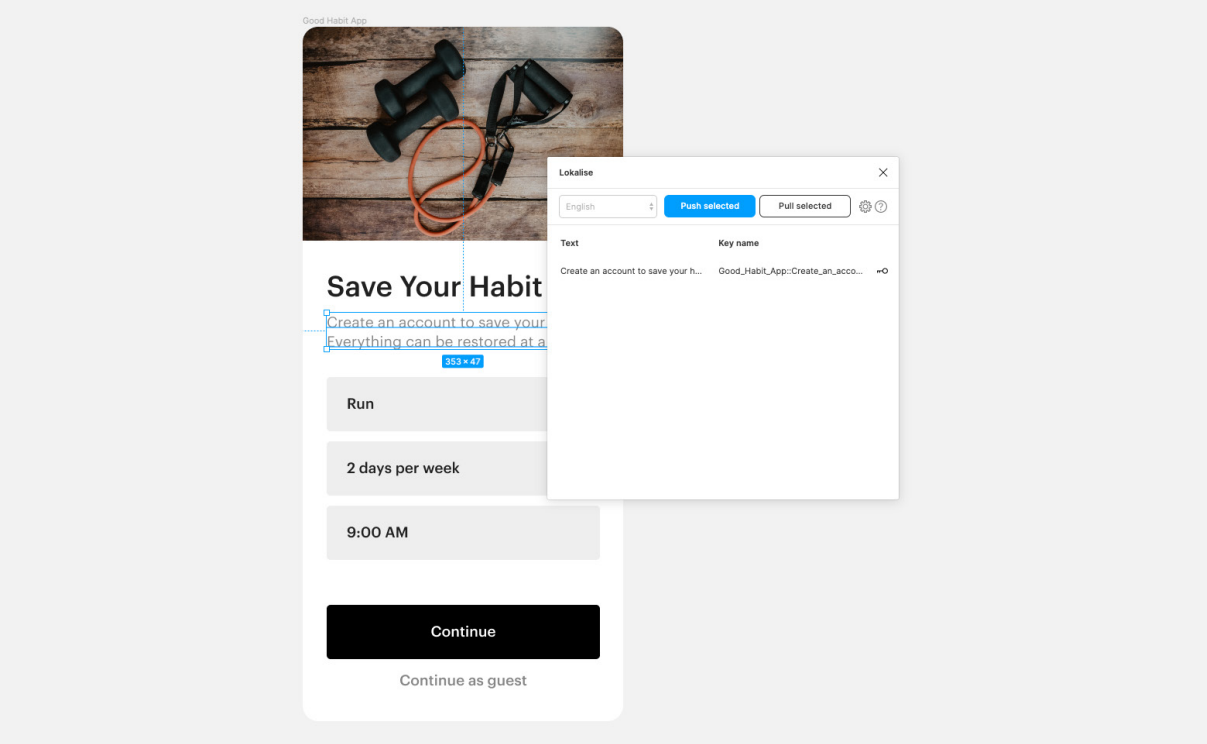
Designers create their prototypes and mockups in Figma or Adobe XD, populate them with different languages, and are able to check how the design will look with different translations early in the process. That means your designer can now see if the design has to be altered to suit different locales **before a single line of code is written**.

This improved localization workflow allows your team to:

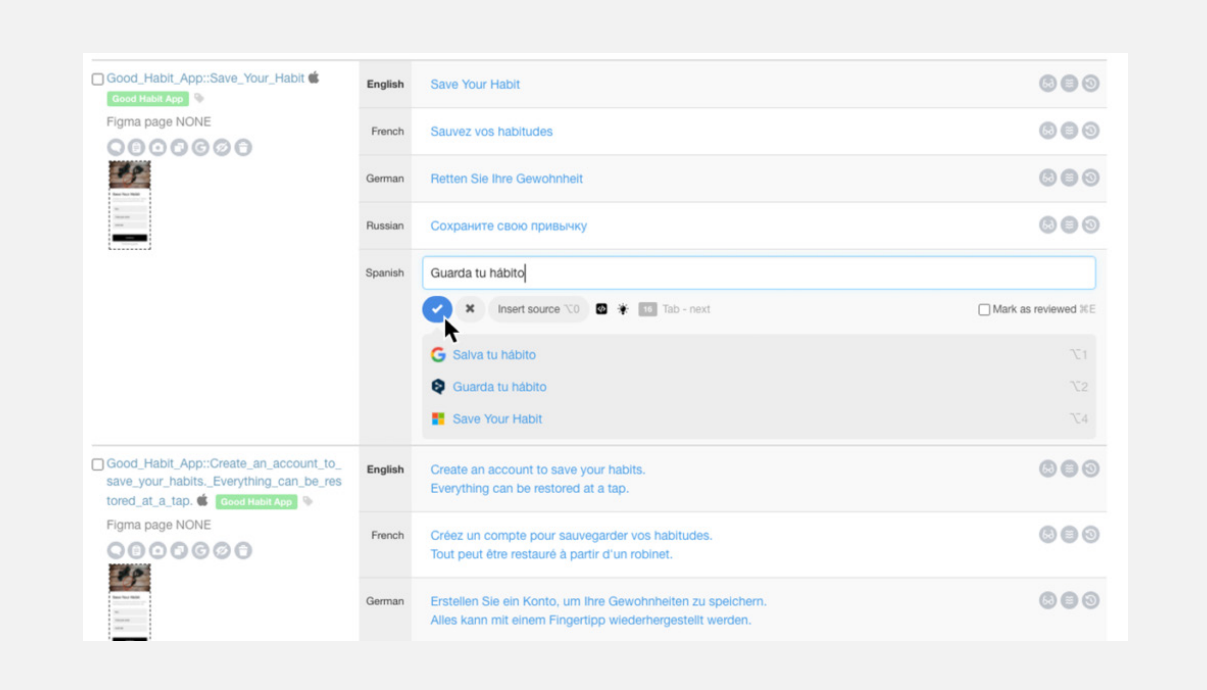
- Catch design breaks at an early stage of your product development process.
- Get product feedback early in your development process, both from linguists and/or user tests in multiple languages.
- Lower the risk of localization errors in your product.
- Eliminate the seemingly endless back and forth between designers, developers, managers, and translators.

Here's a simplified overview of the process:

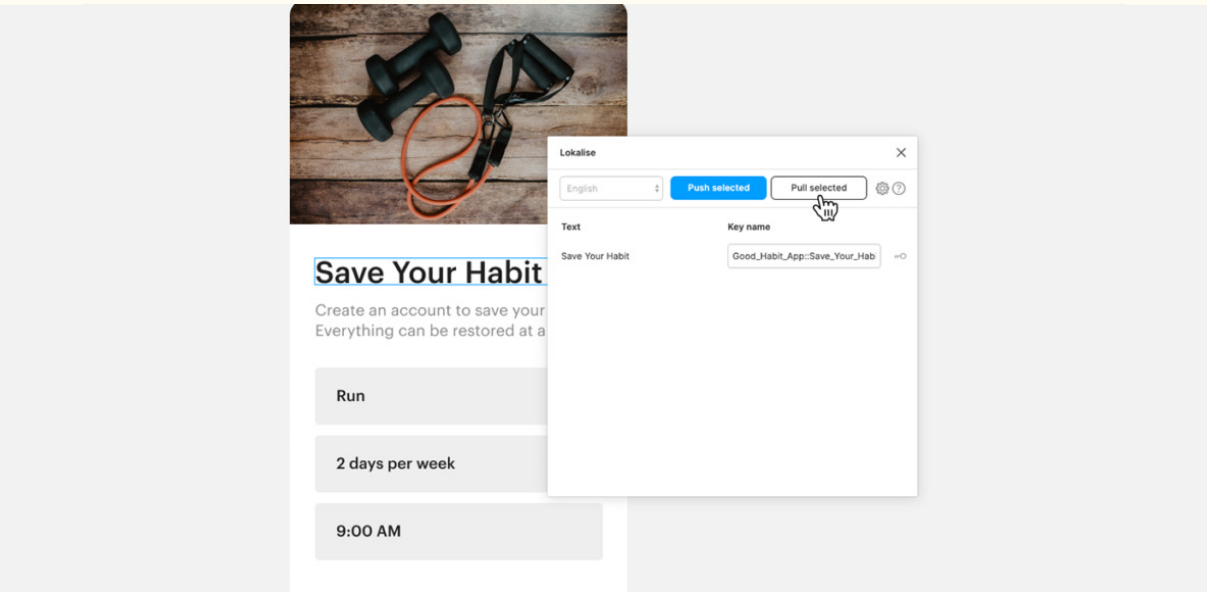
1. Push texts from Figma or Adobe XD to Lokalise (with screenshots attached).



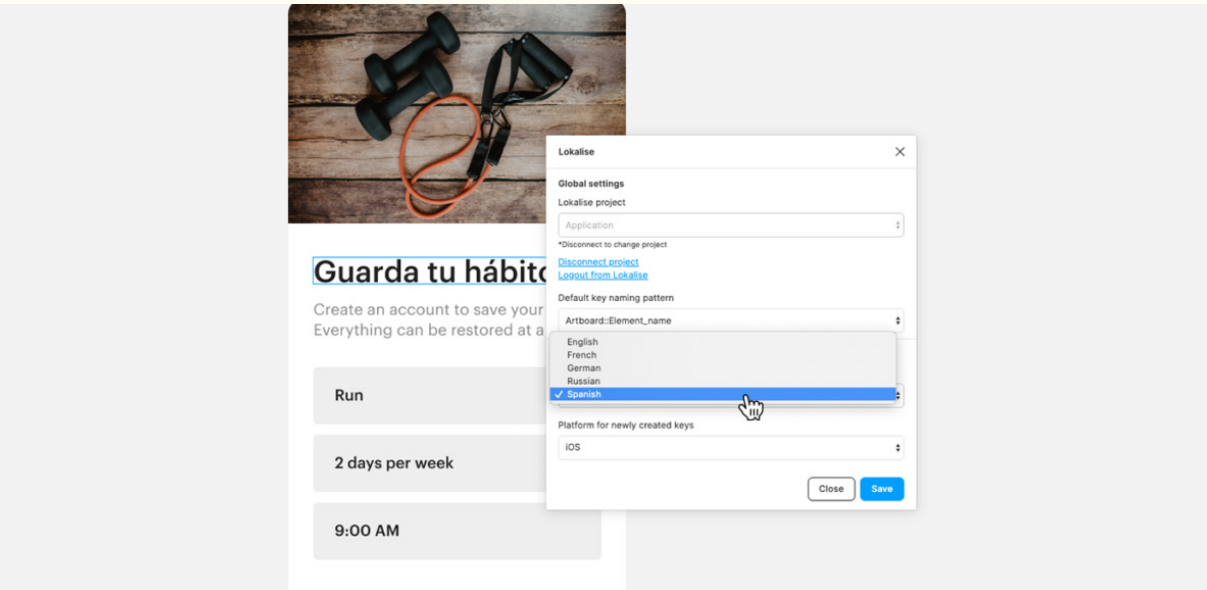
2. Translators create translations in Lokalise.



3. The designer can “pull” translations from Lokalise back to Figma or Adobe XD.



4. The designer can switch between languages to view how the different language texts appear in the designs.



Takeaway: Fixing an already completed software design because the translated content does not fit the interface may take more time and effort from developers than simply integrating localization from the very beginning.

09

Providing context to translators

The last localization issue is providing context to translators.

There are thousands of words and phrases that are often provided out of context. As a result, they are often impossible to translate without knowing some background information.

For example, consider English words like go, pool, mine, and draft. Words like these are homonyms — words that, depending on the context, have completely different meanings. Pool could refer to a chlorinated body of water or a game of billiards. Draft could refer to beer from a keg or a gust of wind.

Translators require context in order to deliver quality results. They need to understand the specifics of your app, where and how the given texts are utilized, what their purpose and functions are, and so on.

Here's how we at Lokalise approached this. Key comments and a built-in chat function are the features that enable users to understand context and stay in the loop. Adding key comments is simple and requires just a couple of clicks. Translators can then read them and ask additional questions in the project chat.

In addition, devs and designers can upload screenshots and link them to translation keys, and offer additional convenient features such as automatic key detection. This feature tries to recognise the text on an uploaded image and link these portions of text to existing keys. As the old saying goes: a picture is worth a thousand words. Therefore, showing how and where exactly the text is to be displayed comes in handy.

To provide additional context, developers should be able to set up a web in-context

editor. With this editor, translators can manage texts directly on the webpage; far better than squinting their eyes at spreadsheets.

Getting started with the LiveJS editor is easy. First of all, include LiveJS script in your page:

```
<script>
    window.LOKALISE_CONFIG = {
      projectId: “18302045592fa799a35d20.15846093”,
      locale: “en”
    };
    (function () {
      var a = document.createElement(“script”);a.type = “text/
javascript”;a.async = !0;
      a.src = [“https://app.lokalise.com/live-js/script.min.js?”, (new
Date).getTime()].join(“”);
      document.body.appendChild(a)
    })();
</script>
```

Next, expose key names in the HTML, because the editor must understand what text element corresponds to what key. This is achieved by adding data-lokalise and data-key attributes:

```
<span data-lokalise data-key=”index.hero.title”>Translation platform for
developers</span>
```

And that’s it! Now when opening the webpage, you’ll see the in-context editor and will be able to use it to perform translations. After hitting “Save”, all the changes will be forwarded to the specified Lokalise project.

Takeaway: The context shared with a translator in screenshots, descriptions, or comments on keys, helps significantly improve translation quality and eliminates back and forths between the teams.

Check all relevant boxes for your localization software

As you can see from what has been discussed above, the list of localization pains for developers is quite lengthy. Solving these issues will not only enable developers to get back to their main tasks, but it will also positively impact the speed of release cycles. When you enable developers to set up a process once and just forget about it, it leads to better productivity across different teams, faster time to market, and cost optimisation.

This is exactly the kind of environment we created in Lokalise.

Lokalise is localization software that aims to eliminate the hassle of localization by providing tools to automate, integrate and better manage your translations. Moreover, this software was created by developers, for developers. If you're curious to find out more about how it can help your localization process, schedule a custom demo with one of our specialists.


lokalise