Abhinav Kumar
abhinav6722cool@gmail.com
+918292459672

Internship Project
Exposys Data Labs

Language Translator App
Project Report

—

Name : Language translator app using PHP

Table of Content

Abstract

Introduction

Existing Method

Proposed method with Architecture

Prepare PHP for translations

1. Check PHP modules
2. Set language
3. Set the translation files
4. Set text domain
5. Add text markup

Methodology

Create translation files

1. Extract strings with xgettext
2. Create initial PO files with msginit
3. Translate the PO files
4. Convert PO to MO files using msgfmt
5. Enjoy
6. Managing changes with msgmerge

Dependencies

Implementation

Conclusion

Abstract

Language translator is a php based web app which is used to translate different languages. Unlike other applications it has certain advantages like it can be easily accessed by everyone having internet access. It detects the language automatically entered by the user and can be converted into any of the desired languages. With a beautiful and simple UI it is very handy to use.

Introduction

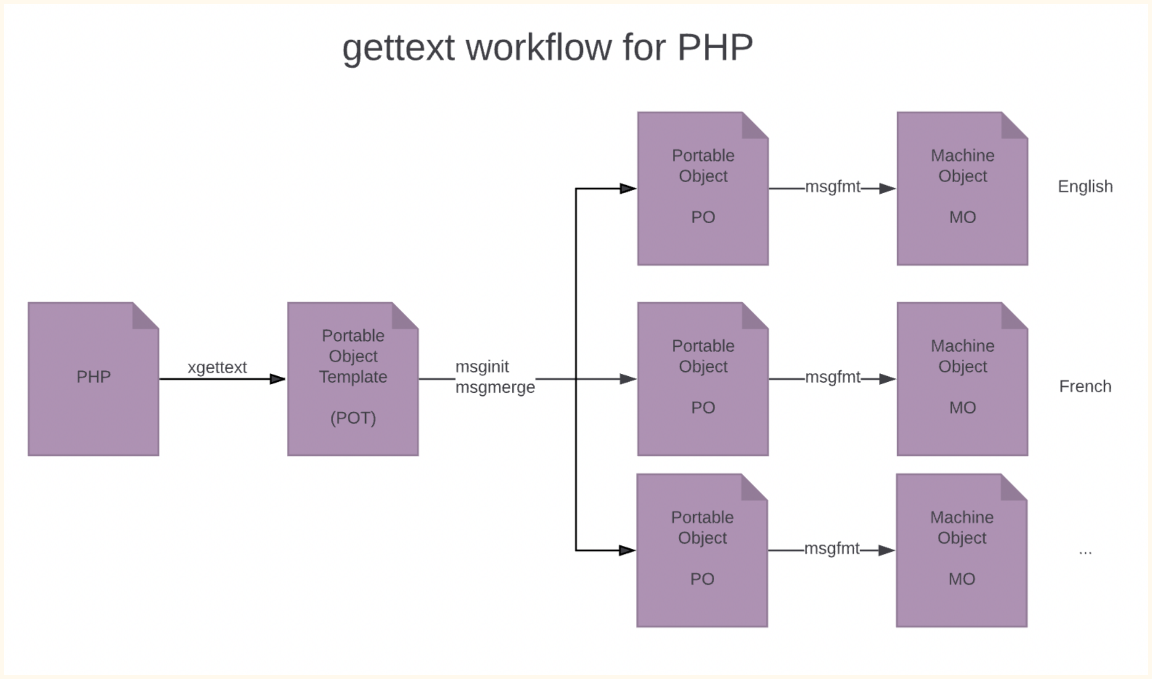
The task was to create a language translator app or web application that can translate from English to any language. There were various approaches to try out this project but since I am familiar with php, html, css, js so I used these languages to create a translator app. See the website live here.

Existing Method

There were various existing methods available which can be used for translation but our main motive was to create something that is completely user friendly along with beautiful UI and is achievable.

Proposed method with Architecture

The following image shows the translation workflow for the app. The basic steps are:



1. Extract messages to translate with xgettext, stored as Portable Object Template file (.pot)
2. Use msginit and msgmerge to create a Portable Object file (.po) for each language
3. Convert the PO files into optimized Machine Object Files (.mo) using msgfmt

Prepare PHP for translations

Important notice: Before you start using gettext check if your development and production server both support it:

- PHP modules gettext and mbstring must be installed on your server
- the locales you want to support must be installed The next 2 sections explain how you can check the requirements:

1. Check PHP modules

The translation functionality in PHP requires the modules gettext and mbstring. Call phpInfo() from a script to see if these modules are both activated. If you can't see them both install and activate them in the php.ini file.

2. Set language

Start by setting the language you want to display. It does not matter where you do that: It's just vital that you set the language before you request the first translated string.

```
setlocale(LC_ALL, 'de_DE.UTF-8');
```

The gettext module in PHP only supports languages and language identifiers that are installed on your server.

Check the available locales by calling

```
locale -a
```

on the command line of your server (e.g., by logging in with SSH).

You can install missing locales with

```
locale-gen <language>
```

The command may require admin privileges and maybe not work on a hosted shared server. If the language you want to use is not available, you have to ask your admin or switch to a different solution.

3. Set the translation files

Tell PHP the location of your translations:

bindtextdomain("myapp", "locale");

- **myapp** is called “text domain” and corresponds to the translation file name. You can call bindtextdomain() for multiple files and switch between them later.
- **locale** is the directory from which the translation files will be loaded. Its path is relative to the PHP file. In our example PHP will try to load the following translation file:

locale/de_DE.UTF-8/LC_MESSAGES/myapp.mo

LC_MESSAGES is a predefined folder name, it is mandatory.

de_DE.UTF-8 is the locale identifier you’ve passed to setlocale() before. In the folder name, country code, and the encoding are optional. The file loader will also try to find these translation files:

locale/de_DE/LC_MESSAGES/myapp.mo

locale/de/LC_MESSAGES/myapp.mo

4. Set text domain

Select the text domain you want to use for translation lookups for all following _() calls:

textdomain("myapp");

The passed name must be bound to a translation file before with **bindtextdomain()**, see above.

5. Add text markup

Now you can wrap all translatable strings with the _() function. This function is an alias for the function gettext. It replaces the passed string with the translation of the language you set before with setlocale():

```
<h1>
   <?="<"?>?php echo _("Translating PHP pages with gettext")?>
</h1>
```

The _() function is also used to extract translatable strings from your source code automatically.

Methodology

Create translation files

1. Extract strings with xgettext

With the xgettext tool you can extract all translatable strings from the PHP source code, store them in a PO template file with file extension .pot. Don't edit the POT file - xgettext overwrites it each time you start it.

```
xgettext --add-comments *.php myapp.pot
```

With the –add-comments option comment blocks preceding the _(...) expression are copied from the source code to the .pot file. With such comments, you can give a hint to the translator how he should translate the text.

2. Create initial PO files with msginit

The POT file is the template file used to create a new PO file for each target language. Place them in directories as mentioned above:

```
msginit --locale de --input myapp.pot --output locale/de/LC_MESSAGES/myapp.po
```

```
msginit --locale fr --input myapp.pot --output  locale/fr/LC_MESSAGES/myapp.po
```

This step is only needed once when you set up your project. As msginit overwrites an existing PO file, you shouldn't call it if you have already PO files containing translations. In this case use msgmerge as described later.

3. Translate the PO files

Now you can start translating the PO files. You can use a text editor or a special translation editor for that. If your translations contain non-ASCII characters, change the charset in the header from ASCII to UTF-8:

```
"Content-Type: text/plain; charset=UTF-8\n"
```

4. Convert PO to MO files using msgfmt

You next have to convert the PO files into “Message Object” files (with the extension .mo) which can be loaded on the server:

```
cd locale/de/LC_MESSAGES/
```

```
msgfmt myapp.po --output-file=myapp.mo
```

5. Enjoy

Start your PHP script and enjoy its translated output. Keep in mind that gettext caches the translation data. If you add/remove/update MO files you might have to restart/reload your web server.

6. Managing changes with msgmerge

If you're updating your PHP code, you might change translatable texts, too. This requires an update of the PO/MO files. There's, of course, a gettext tool to simplify this task, it's called msgmerge. First, you have to extract the translatable strings from the new PHP sources using xgettext. This overwrites your old POT file with a new one. Then you can merge the changes into the language-specific PO files:

xgettext –add-comments *.php myapp.pot msgmerge –update locale/de/LC_MESSAGES/myapp.po myapp.pot … translate new / changed texts in myapp.po …

msgfmt locale/de/LC_MESSAGES/myapp.po –output-file=locale/de/LC_MESSAGES/myapp.mo msgmerge doesn't delete any translation: If a string is no longer found in the source code, it is marked as deleted in the PO file:

#~ msgid "Draft"

#~ msgstr "Entwurf"

New strings are added to the PO file, for unchanged strings the source code line number, and the extracted comment are updated, if necessary.

msgmerge also tries to detect changed strings using a fuzzy string comparison. If only a small part of the string has changed the new string is saved together with the old translation in the PO file, and the entry is marked as "fuzzy".

Example: If you change "Translation of PHP pages with gettext" to "Translate your PHP pages with gettext" you will get this PO file entry. The translation is outdated and should be updated in the next step:

```
#: index.php:17

#, fuzzy

msgid "Translate your PHP pages with gettext"

msgstr "Übersetzen von PHP-Seiten mit gettext"
```

If **msgmerge** cannot detect a similarity between old and new string, the old one is marked as deleted, and the new one is added as a new entry, without translation.

Dependencies

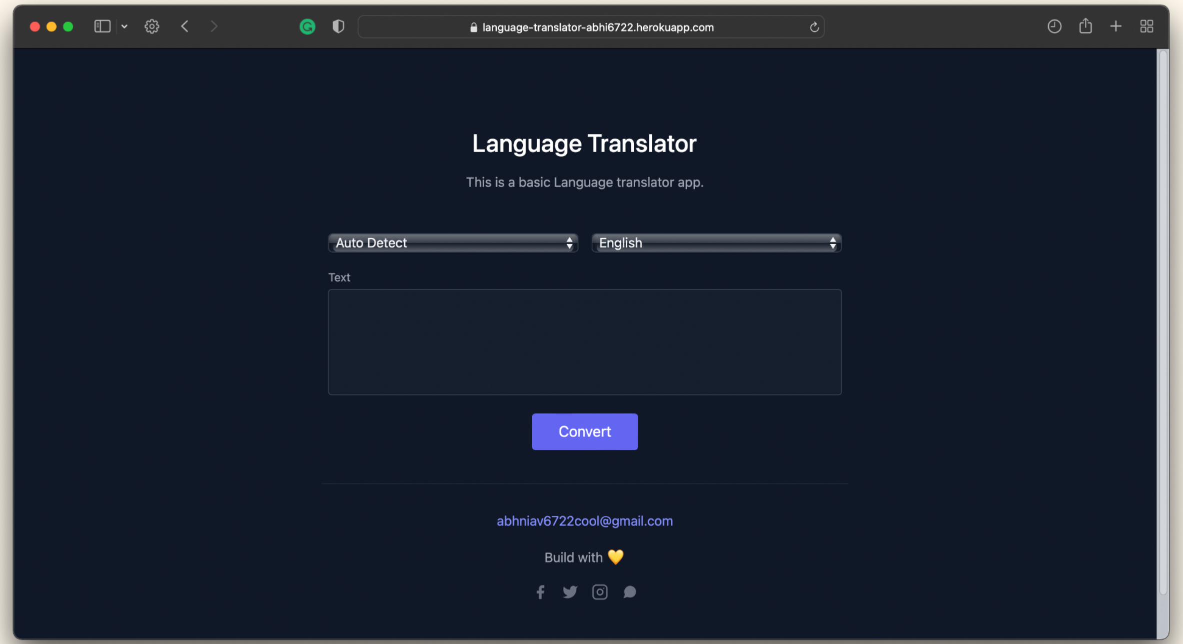
The project has following dependencies

- php (8.1.0)
- ext-mbstring (bundled with php)
- composer (2.1.14)
- apache (2.4.51)

- nginx (1.20.2)

Implementation

Since the main objective from the start was to make it easy to access from anywhere, deploying the project was an important phase. First the project was successfully uploaded on Github and then it was deployed to Heroku.


language-translator-abhi6722.herokuapp.com
Language Translator
This is a basic Language translator app.
Auto Detect
English
Text
Convert
abhniav6722cool@gmail.com
Build with 💛

Conclusion

The Language Translator web app is successfully created and deployed to Heroku here. The main objectives were also fulfilled as the UI is easy to use and also Beautiful along with various languages provided.

- Link to Application : https://language-translator-abhi6722.herokuapp.com
- Link to Repository : https://github.com/Abhi6722/language-translator
- Link to the Presentation : https://docs.google.com/presentation/d/1nbB6OkQARDjqiui5r7AC6tvbRQRy-od2_zzsz1DYUd0/edit?usp=sharing