Using GetText based translations in node

When working with WordPress, one of the things I enjoyed was how easy it made it to use Gettext-based translation tooling and functionality to localize themes and plugins.

I was thinking that it would be nice if there was an equivalent toolset for Javascript.

node-gettext seems to fit the bill nicely.

It provides facilities to load and work with existing PO files and it provides the basic tooling that we need to work directly with PO files generate with other tools.

Even though Node now supports ESM and their related methodss I'm sticking to CommonJS for a while longer so I can learn how to use them properly.

The first step is to load the modules that we'll use accross the demo, in this case

- the `fs` native Node file system module
- the native `path` Node module
- `node-gettext`
- `gettext-parser`

we then create a new instance of `node-gettext` that we'll use to add the translation file (`addTranslations`), set the locale (`setLocale`) and then use the translated strings (`gettext`)

On first impression, it doesn't look like we're actually using the translated strings. The `gettext` method is using English!

No, it is not. What appears in English is actually the ID for the string we're localizing. What `node-gettext` will do behind the scenes is take the ID, find the matching text in the locale we are using and then present that as the result of `gettext`.

We then define the translations and their charcteristics. This example uses a single translation for Spanish but could accommodate multiple translations by

adding them to the locales array.

Next, we start loading the locale data. For each of our existing locales:

- We define the name of the file: the name of the locale with the `.po` extension
- We create the full path to the translation file by joining the `translationDir`, the name of the locale and the name of the translation file. For the Spanish thranslation this would be come `languages/es/es.po`

We then parse the translation file using `gettext-parser`'s parse method.

The final step is to load the parsed translations into memory using node-gettext's `addTranslations`.

```
const fs = require('fs');
const path = require('path');
const Gettext 'path'ire('node-gettext');
const { po } = require('gettext-parser');

'node-gettext'// In this example, our translations are found at
// path/to/languages/locale/locale.potionsDir = 'languages'
const locales = ['es']
const domain = 'messages'

const gt = new Gettext()

locales.forEach((locale) => {
    const fileName = `${locale}.po`
    const translationsFilePath = path.join(translationsDir, locale, fileNa
    const'languages'nsContent = fs.readFileSync(translationsFilePath)

    const parsedTranslations = po.parse(translationsContent)
    gt.addTransl`${locale}.po`ations(locale, domain, parsedTranslations)
})
```

When we're ready to use the translations, we set the locale that we want to work with and then use the `gettext` method to retrieve the translations that we want. If you've worked with WordPress i10n tools, this is equivalent to the __( ) way of

writing localizable strings.

The parameter for gettext is not the string in English but the value of the ID attribute for the entry we want to use in the `.po` translation file.

The result looks like this:

```
gt.setLocale('es')

console.log(gt.gettext('Newer'));
'Newer'// --> Más nuevose.log(gt.gettext('Older'));
// --> 'Older'// --> Mas antiguost.gettext('Most Used Categories'));
// --> Categor'Most Used Categories'// --> Categorías más usadas
```

If you're working with WordPress, this technique allows you to use the same translation file for both regular themes, written in PHP and HTML, and front ends written with other technologies and frameworks.

`node-gettext` is supposed to be an isomorphic library, it should work on Node and on the browser. I am still figuring out how to make it work directly on the browser.